

ICPC REFERENCE

Escuela Superior de Cómputo - IPN

Alberto Silva

Índice

1. Number Theory	2	1.4.6. Cofactors Matrix	12
1.1. Aritmetica modular	2	1.4.7. Matriz inversa	12
1.1.1. Inverso modular	2	1.4.8. Adjoint Matrix	12
1.1.2. Linear Congruence Equation	2	1.4.9. Recurrencias lineales	12
1.1.3. Factorial modulo p	2	1.4.10. Kirchhoff Matrix Tree Theorem	13
1.1.4. Chinese Remainder Theorem	2	1.5. Combinatorics	14
1.2. Cribas y primos	2	1.5.1. Binomial coefficients	14
1.2.1. Criba de eratostenes	2	2. Strings	16
1.2.2. Criba de factor primo más pequeño	3	2.1. Trie	16
1.2.3. Criba de la función φ de Euler	3	2.2. Suffix array and LCP	17
1.2.4. Criba de la función μ	3	2.3. Suffix Tree	17
1.2.5. Triángulo de Pascal	3	2.4. Suffix automaton	19
1.2.6. Criba de primos lineal	3	2.5. Aho corasick	19
1.2.7. Block sieve	3	2.6. Z function	19
1.2.8. Prime factors of $n!$	4	2.7. Knuth morris pratt	19
1.2.9. Primality test(miller rabin)	4	2.8. Palindromic tree	19
1.2.10. Factorización varios metodos	5	2.9. Manacher	19
1.2.11. Factorizacion usando todos los metodos	7		
1.2.12. Numero de divisores hasta 10^{18}	7		
1.3. Funciones multiplicativas	8		
1.3.1. Función φ de Euler	8		
1.4. Linear Algebra	9		
1.4.1. Struct matrix	9		
1.4.2. Transpuesta	10		
1.4.3. Traza	10		
1.4.4. Gauss System of Linear Equationsn	11		
1.4.5. Gauss Determinant	11		

1. Number Theory

1.1. Aritmetica modular

1.1.1. Inverso modular

```
int inverse(int a, int m){
    int x, y;
    if isPrime(m) return mod_pow(a,m-2,m);
    if(gcd( a, m, x, y )  $\neq$  1) return 0;
    return (x%m + m) % m;
```

```
*/
vector<lli> allinverse(lli p){
    vector<lli> ans(p);
    ans[1] = 1;
    for(lli i = 2; i<p; i++){
        ans[i] = p-(p/i)*ans[p%i]%p;
    }
    return ans;
}
```

1.1.2. Linear Congruence Equation

```
bool find_any_solution(int a, int b, int c, int &x0, int
    &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g)
        return false;

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
```

1.1.3. Factorial modulo p

```
vector<int> rem;
int CRT() {
    int prod = 1;
    for (int i = 0; i < nums.size(); i++)
```

```
    prod *= nums[i];
    int result = 0;
    for (int i = 0; i < nums.size(); i++) {
        int pp = prod / nums[i];
        result += rem[i] * inverse(pp, nums[i]) * pp;
    }
```

1.1.4. Chinese Remainder Theorem

```
inline lli normalize(lli x, lli mod) { x %= mod; if (x < 0)
    <math>\rightarrow</math> x += mod; return x; }
vector<int> a;
vector<int> n;
lli LCM;
lli CRT(lli &ans){
    int t = a.size();
    ans = a[0];
    LCM = n[0];
    for(int i = 1; i < t; i++){
        int x1,d= gcd(LCM, n[i],x1,d);
        if((a[i] - ans) % d  $\neq$  0) return 0;
        ans = normalize(ans + x1 * (a[i] - ans) / d % (n[i]
            <math>\rightarrow</math> / d) * LCM, LCM * n[i] / d);
        LCM = lcm(LCM, n[i]); // you can save time by
            <math>\rightarrow</math> replacing above LCM * n[i] / d by LCM = LCM *
            <math>\rightarrow</math> n[i] / d
    }
    return 1;
}
```

1.2. Cribas y primos

1.2.1. Criba de eratostenes

```
vector<int> Criba(int n) {
    int raiz = sqrt(n);
    vector<int> criba(n + 1);
    for (int i = 4; i  $\leq$  n; i += 2)
        criba[i] = 2;
    for (int i = 3; i  $\leq$  raiz; i += 2)
        if (!criba[i])
            for (int j = i * i; j  $\leq$  n; j += i)
                if (!criba[j]) criba[j] = i;
```

```

    return criba;
}

```

1.2.2. Criba de factor primo más pequeño

```

vector<int> lowestPrime;
void lowestPrimeSieve(int n){
    lowestPrime.resize(n + 1, 1);
    lowestPrime[0] = lowestPrime[1] = 0;
    for(int i = 2; i ≤ n; ++i)
        lowestPrime[i] = (i & 1 ? i : 2);
    int limit = sqrt(n);
    for(int i = 3; i ≤ limit; i += 2)
        if(lowestPrime[i] == i)
            for(int j = i * i; j ≤ n; j += 2 * i)
                if(lowestPrime[j] == j) lowestPrime[j] = i;
}

```

1.2.3. Criba de la función φ de Euler

```

vector<int> Phi;
void phiSieve(int n){
    Phi.resize(n + 1);
    for(int i = 1; i ≤ n; ++i)
        Phi[i] = i;
    for(int i = 2; i ≤ n; ++i)
        if(Phi[i] == i)
            for(int j = i; j ≤ n; j += i)
                Phi[j] -= Phi[j] / i;
}

```

1.2.4. Criba de la función μ

```

vector<int> Mu;
void muSieve(int n){
    Mu.resize(n + 1, -1);
    Mu[0] = 0, Mu[1] = 1;
    for(int i = 2; i ≤ n; ++i)
        if(Mu[i])
            for(int j = 2*i; j ≤ n; j += i)
                Mu[j] -= Mu[i];
}

```

1.2.5. Triángulo de Pascal

```

vector<vector<lli>> Ncr;
void ncrSieve(lli n){
    Ncr.resize(n + 1);
    Ncr[0] = {1};
    for(lli i = 1; i ≤ n; ++i){
        Ncr[i].resize(i + 1);
        Ncr[i][0] = Ncr[i][i] = 1;
        for(lli j = 1; j ≤ i / 2; j++)
            Ncr[i][i - j] = Ncr[i][j] = Ncr[i - 1][j - 1] +
                Ncr[i - 1][j];
    }
}

```

1.2.6. Criba de primos lineal

```

const int N = 100000000;
int lp[N+1];
vector<int> primes;
void criba(){
    for (int i=2; i≤N; ++i) {
        if (lp[i] == 0) {
            lp[i] = i;
            primes.push_back (i);
        }
        for (int j=0; j<(int)primes.size() &&
            lp[i * primes[j]] == 0; ++j)
            lp[i * primes[j]] = primes[j];
    }
}

```

1.2.7. Block sieve

```

int count_primes(int n) {
    const int S = 10000;
    vector<int> primes;
    int nsqrt = sqrt(n);
    vector<char> is_prime(nsqrt + 1, true);
    for (int i = 2; i ≤ nsqrt; i++) {
        if (is_prime[i]) {

```

```

        primes.push_back(i);
        for (int j = i * i; j ≤ nsqrt; j += i)
            is_prime[j] = false;
    }
}
int result = 0;
vector<char> block(S);
for (int k = 0; k * S ≤ n; k++) {
    fill(block.begin(), block.end(), true);
    int start = k * S;
    for (int p : primes) {
        int start_idx = (start + p - 1) / p;
        int j = max(start_idx, p) * p - start;
        for (; j < S; j += p)
            block[j] = false;
    }
    if (k == 0)
        block[0] = block[1] = false;
    for (int i = 0; i < S && start + i ≤ n; i++) {
        if (block[i])
            result++;
    }
}
return result;
}

```

1.2.8. Prime factors of $n!$

if p is prime the highest power p^k of p that divides $n!$ is given by

$$k = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \left\lfloor \frac{n}{p^3} \right\rfloor + \dots$$

1.2.9. Primality test(miller rabin)

```

lli random(lli a, lli b) {
    lli intervallLength = b - a + 1;
    int neededSteps = 0;
    lli base = RAND_MAX + 1LL;
    while(intervallLength > 0){
        intervallLength /= base;
        neededSteps++;
    }
}

```

```

    }
    intervallLength = b - a + 1;
    lli result = 0;
    for(int stepsDone = 0; stepsDone < neededSteps;
        → stepsDone++){
        result = (result * base + rand());
    }
    result %= intervallLength;
    if(result < 0) result += intervallLength;
    return result + a;
}

bool witness(lli a, lli n) {
    lli u = n-1;
    int t = 0;
    while (u % 2 == 0) {
        t++;
        u /= 2;
    }
    lli next = mod_pow(a, u, n);
    if(next == 1) return false;
    lli last;
    for(int i = 0; i < t; i++) {
        last = next;
        next = mod_mult(last, last, n); //(last * last) % n;
        if (next == 1){
            return last ≠ n - 1;
        }
    }
    return next ≠ 1;
}

bool isPrime(lli n, int s) {
    if (n ≤ 1) return false;
    if (n == 2) return true;
    if (n % 2 == 0) return false;
    for(int i = 0; i < s; i++) {
        lli a = random(1, n-1);
        if (witness(a, n)) return false;
    }
}

```

```

    return true;
}

```

1.2.10. Factorización varios metodos

```

map<lli, lli> fact;
void trial_division4(lli n) {
    for (lli d : primes) {
        if (d * d > n)
            break;
        while (n % d == 0) {
            fact[d]++;
            n /= d;
        }
    }
}

void trial_division2(lli n) {
    while (n % 2 == 0) {
        fact[2]++;
        n /= 2;
    }
    for (long long d = 3; d * d ≤ n; d += 2) {
        while (n % d == 0) {
            fact[d]++;
            n /= d;
        }
    }
    if (n > 1)
        fact[n]++;
}

/*
Pollard Method p-1
*/
lli pollard_p_1(lli n){
    int b = 13;
    int q[] = {2, 3, 5, 7, 11, 13};
    lli a = 5% n;
    for (int j = 0; j < 10; j++){
        while (__gcd(a, n) != 1){
            mod_mult (a, a, n);
            a += 3;
            a %= n;
        }
    }
}

```

```

    }
    for (int i = 0; i < 6; i++){
        int qq = q[i];
        int e = floor(log((double) b) / log((double) qq));
        lli aa = mod_pow(a, mod_pow (qq, e, n), n);
        if (aa == 0)
            continue;
        lli g = __gcd (aa-1, n);
        if (1 < g && g < n)
            return g;
    }
}

return 1;
}

/*
Pollard rho
*/
lli pollard_rho (lli n, unsigned iterations_count =
↳ 100000){
    lli b0 = rand ()% n, b1 = b0, g;
    mod_mult (b1, b1, n);
    if (++b1 == n)
        b1 = 0;
    g = __gcd(abs(b1 - b0), n);
    for (unsigned count = 0; count < iterations_count && (g ==
↳ 1 || g == n); count++){
        mod_mult (b0, b0, n);
        if (++b0 == n)
            b0 = 0;
        mod_mult (b1, b1, n);
        ++b1;
        mod_mult (b1, b1, n);
        if (++b1 == n)
            b1 = 0;
        g = __gcd(abs(b1 - b0), n);
    }
    return g;
}

lli pollard_bent (lli n, unsigned iterations_count = 19){
    lli b0 = rand ()% n,

```

```

    b1 = (b0 * b0 + 2) % n;
    a = b1;
    for (unsigned iteration = 0, series_len = 1; iteration
    ↪ < iterations_count; iteration ++, series_len *= 2){
        lli g = __gcd(b1-b0, n);
        for (unsigned len = 0; len < series_len && (g == 1 && g
        ↪ == n); len ++){
            b1 = (b1 * b1 + 2) % n;
            g = __gcd(abs (b1-b0), n);
        }
        b0 = a;
        a = b1;
        if (g != 1 && g != n)
            return g;
    }
    return 1;
}
/*
    Pollard monte Carlo
*/
lli pollard_monte_carlo (lli n, unsigned m = 100){
    lli b = rand () % (m-2) + 2;
    lli g = 1;
    for (int i = 0; i < 100 && g == 1; i++){
        lli cur = primes[i];
        while (cur ≤ n)
            cur *= primes[i];
        cur /= primes[i];
        b = mod_pow (b, cur, n);
        g = __gcd(abs (b-1), n);
        if (g == n)
            g = 1;
    }
    return g;
}
lli prime_div_trivial (lli n){
    if (n == 2 || n == 3)
        return 1;
    if (n < 2)
        return 0;
    if (!n & 1)
        return 2;

```

```

    lli pi;
    for (auto p:primes){
        if (p*p > n)
            break;
        else
            if (n % p == 0)
                return p;
    }
    if (n < 1000*10000)
        return 1;
    return 0;
}

lli ferma (lli n){
    lli x = floor(sqrt((double)n)), y = 0, r = x * x - y * y -
    ↪ n;
    for (;;)
        if (r == 0)
            return x != y? x*y: x + y;
        else
            if (r > 0){
                r -= y + y + 1;
                ++y;
            }
            else{
                r += x + x + 1;
                ++x;
            }
    }
    lli mult(lli a, lli b, lli mod) {
        return (lli)a * b % mod;
    }

    lli f(lli x, lli c, lli mod) {
        return (mult(x, x, mod) + c) % mod;
    }
    lli brent(lli n, lli x0=2, lli c=1) {
        lli x = x0;
        lli g = 1;
        lli q = 1;
        lli xs, y;

```

```

int m = 128;
int l = 1;
while (g == 1) {
    y = x;
    for (int i = 1; i < l; i++)
        x = f(x, c, n);
    int k = 0;
    while (k < l && g == 1) {
        xs = x;
        for (int i = 0; i < m && i < l - k; i++) {
            x = f(x, c, n);
            q = mult(q, abs(y - x), n);
        }
        g = __gcd(q, n);
        k += m;
    }
    l *= 2;
}
if (g == n) {
    do {
        xs = f(xs, c, n);
        g = __gcd(abs(xs - y), n);
    } while (g == 1);
}
return g;
}

```

1.2.11. Factorizacion usando todos los metodos

```

void factorize (lli n){
    if (isPrime(n,20))
        fact[n]++;
    else{
        if (n < 1000 * 1000){
            lli div = prime_div_trivial(n);
            fact[div]++;
            factorize(n / div);
        }
        else{
            lli div;
            // Pollard's fast algorithms come first
            div = pollard_monte_carlo(n);

```

```

        if (div == 1)
            div = brent(n);
        if (div == 1)
            div = pollard_rho (n), cout<<"USE
            ↪ POLLAR_RHO\n";
        if (div == 1)
            div = pollard_p_1 (n), cout<<"USE
            ↪ POLLARD_P_1\n";
        if (div == 1)
            div = pollard_bent (n), cout<<"USE
            ↪ POLLARD_BENT\n";
        if (div == 1)
            div = ferma (n);
        // recursively process the found factors
        factorize (div);
        factorize (n / div);
    }
}
}

```

1.2.12. Numero de divisores hasta 10^{18}

```

bool isSquare(lli val){
    lli lo = 1, hi = val;
    while(lo ≤ hi){
        lli mid = lo + (hi - lo) / 2;
        lli tmp = (val / mid) / mid; // be careful with
        ↪ overflows!!
        if(tmp == 0)hi = mid - 1;
        else if(mid * mid == val)return true;
        else if(mid * mid < val)lo = mid + 1;
    }
    return false;
}

lli countDivisors(lli n) {
    lli ans = 1;
    for(int i = 0; i < primes.size(); i++){
        if(n == 1)break;
        int p = primes[i];
        if(n % p == 0){ // checks whether p is a divisor of n
            int num = 0;
            while(n % p == 0){

```

```

    n /= p;
    ++num;
}
// p^num divides initial n but p^(num + 1) does not
↪ divide initial val
// ⇒ p can be taken 0 to num times ⇒ num + 1
↪ possibilities!!
ans *= num + 1;
}
}
if(n == 1) return ans; // first case
else if(isPrime(n, 20)) return ans * 2; // second case
else if(isSquare(n)) return ans * 3; // third case but
↪ with p = q
else return ans * 4; // third case with p ≠ q
}

```

1.3. Funciones multiplicativas

1.3.1. Función φ de Euler

```

//-----PHI de euler-----//
int phi(int n) {
    int result = n;
    for (int i = 2; i * i ≤ n; i++) {
        if(n % i == 0) {
            while(n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if(n > 1)
        result -= result / n;
    return result;
}

```

The most famous and important property of Euler's totient function is expressed in **Euler's theorem**:

$$\alpha^{\phi(m)} \equiv 1 \pmod{m} \quad (1)$$

if α and m are relative prime.

In the particular case when m is prime, Euler's theorem turns into **Fermat's little theorem**:

$$\alpha^{m-1} \equiv 1 \pmod{m} \quad (2)$$

$$\alpha^n \equiv \alpha^{n \bmod \phi(m)} \pmod{m} \quad (3)$$

This allows computing $x^n \bmod m$ for very big n , especially if n is the result of another computation, as it allows to compute n under a modulo.

1.4. Linear Algebra

1.4.1. Struct matrix

```

template <typename T>
struct Matrix {
    vector < vector <T> > A;
    int r,c;
    Matrix(){
        this->r = 0;
        this->c = 0;
    }
    Matrix(int r,int c){
        this->r = r;
        this->c = c;
        A.assign(r , vector <T> (c));
    }

    Matrix(int r,int c,const T &val){
        this->r = r;
        this->c = c;
        A.assign(r , vector <T> (c , val));
    }
    Matrix(int n){
        this->r = this->c = n;
        A.assign(n , vector <T> (n));
        for(int i=0;i<n;i++)
            A[i][i] = (T)1;
    }
    Matrix operator * (const Matrix<T> &B){
        // Matrix <T> C(r,B.c,0);
        // for(int i=0 ; i<r ; i++)
        //     for(int j=0 ; j<B.c ; j++)
        //         for(int k=0 ; k<c ; k++)
        //             C[i][j] = (C[i][j] + ( (long long
        // ↪ )A[i][k] * (long long)B[k][j] ));
        // return C;
        Matrix<T> C(r,B.c,0);
        for(int i = 0;i<r;i++){
            for(int j = 0;j<B.c;j++){
                for(int k = 0;k<c;k++){
                    C[i][j] = (C[i][j] + ((lli)A[i][k] *
                    ↪ (lli)B[k][j]));
                }
            }
        }
    }

```

```

        if(C[i][j] ≥ 8ll*mod*mod)
            C[i][j]%=mod;
    }
}

for(int i = 0;i<r;i++)for(int j =
    ↪ 0;j<c;j++)C[i][j]%=mod;
return C;
}

Matrix operator + (const Matrix<T> &B){
    assert(r == B.r);
    assert(c == B.c);
    Matrix <T> C(r,c,0);
    int i,j;
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            C[i][j] = ((A[i][j] + B[i][j]));
    return C;
}

Matrix operator*(int &c) {
    Matrix<T> C(r, c);
    for(int i = 0; i < r; i++)
        for(int j = 0; j < c; j++)
            C[i][j] = A[i][j] * c;
    return C;
}

Matrix operator - (){
    Matrix <T> C(r,c,0);
    int i,j;
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            C[i][j] = -A[i][j];
    return C;
}

Matrix operator - (const Matrix<T> &B){
    assert(r == B.r);
    assert(c == B.c);
    Matrix <T> C(r,c,0);
    int i,j;
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)

```

```

        C[i][j] = A[i][j] - B[i][j];
    return C;
}
Matrix operator ^ (long long n){
    assert(r == c);
    int i,j;
    Matrix <T> C(r);
    Matrix <T> X(r,c,0);
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            X[i][j] = A[i][j];
    while(n){
        if(n&1)
            C *= X;
        X *= X;
        n /= 2;
    }
    return C;
}
vector<T>& operator [] (int i){
    assert(i < r);
    assert(i ≥ 0);
    return A[i];
}

const vector<T>& operator [] (int i) const{
    assert(i < r);
    assert(i ≥ 0);
    return A[i];
}

friend ostream& operator << (ostream &out, const
↪ Matrix<T> &M){
    for (int i = 0; i < M.r; ++i) {
        for (int j = 0; j < M.c; ++j) {
            out << M[i][j] << " ";
        }
        out << '\n';
    }
    return out;
}

```

```

void operator *= (const Matrix<T> &B){
    (*this) = (*this)*B;
}

void operator += (const Matrix<T> &B){
    (*this) = (*this)+B;
}

void operator -= (const Matrix<T> &B){
    (*this) = (*this)-B;
}

void operator ^= (long long n){
    (*this) = (*this)^n;
}

};

```

1.4.2. Transpuesta

```

Matrix transpose(){
    Matrix <T> C(c,r);
    int i,j;
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            C[j][i] = A[i][j];

    return C;
}

```

1.4.3. Traza

```

T trace(){
    T sum = 0;
    for(int i = 0; i < min(r, c); i++)
        sum += A[i][i];
    return sum;
}

```

1.4.4. Gauss System of Linear Equations

```

int gauss (vector<double> & ans) {
    Matrix<double> Temp(this→r,this→c);
    int n = (int) Temp.A.size();
    int m = (int) Temp[0].size() - 1;
    for(int i = 0; i<n; i++)
        for(int j = 0; j<n; j++)
            Temp[i][j] = (double)A[i][j];

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (fabs (Temp[i][col]) > fabs
                ↳ (Temp[sel][col]))
                sel = i;
        if (fabs (Temp[sel][col]) < EPS)
            continue;
        for (int i=col; i<m; ++i)
            swap (Temp[sel][i], Temp[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i ≠ row) {
                double c = Temp[i][col] /
                    ↳ Temp[row][col];
                for (int j=col; j<m; ++j)
                    Temp[i][j] -= Temp[row][j] * c;
            }
        ++row;
    }
    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] ≠ -1)
            ans[i] = Temp[where[i]][m] /
                ↳ Temp[where[i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * Temp[i][j];
        if (fabs (sum - Temp[i][m]) > EPS)
            return 0;
    }
}

```

```

    }
    for (int i=0; i<m; ++i)
        if (where[i] = -1)
            return INF;
    return 1;
}

```

1.4.5. Gauss Determinant

```

int detGauss(){
    assert(r = c);
    double det = 1;
    Matrix<double> temp(r);
    temp.r = r;
    temp.c = c;
    int n = r;
    for(int i = 0; i<n; i++)
        for(int j = 0; j<n; j++)
            temp[i][j] = (double)A[i][j];
    for (int i=0; i<n; ++i) {
        int k = i;
        for (int j=i+1; j<n; ++j)
            if (fabs (temp[j][i]) > fabs (temp[k][i]))
                k = j;
        if (abs (temp[k][i]) < EPS) {
            det = 0;
            break;
        }
        swap (temp[i], temp[k]);
        if (i ≠ k)
            det = -det;
        det *= temp[i][i];
        for (int j=i+1; j<n; ++j)
            temp[i][j] /= temp[i][i];
        for (int j=0; j<n; ++j)
            if (j ≠ i && abs (temp[j][i]) > EPS)
                for (int k=i+1; k<n; ++k)
                    temp[j][k] -= temp[i][k] *
                        ↳ temp[j][i];
    }
}

```

```

    return (int)det;
}

```

1.4.6. Cofactors Matrix

```

Matrix<T> cofactorMatrix(){
    Matrix<T> C(r, c);
    for(int i = 0; i < c; i++)
        for(int j = 0; j < r; j++)
            C[i][j] = cofactor(i, j);
    return C;
}

```

1.4.7. Matriz inversa

```

bool Inverse(Matrix<double> &inverse){
    if(this->detGauss() == 0) return false;
    int n = A[0].size();
    Matrix<double> temp(n, 2*n);
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++) temp[i][j] = A[i][j];
    Matrix<double> ident(n);
    for(int i = 0; i < n; i++)
        for(int j = n; j < 2*n; j++) temp[i][j] =
            ↪ ident[i][j-n];
    int m = n*2;
    vector<int> where (m, -1);

    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (temp[i][col]) > abs
                ↪ (temp[sel][col]))
                sel = i;
        if (abs (temp[sel][col]) < EPS)
            continue;
        for (int i=col; i<m; ++i)
            swap (temp[sel][i], temp[row][i]);
        where[col] = row;
        double div = temp[row][col];
        for(int i = 0; i < m; i++)

```

```

        if(fabs(temp[row][i])>EPS)temp[row][i]
            ↪ /=div;
        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = temp[i][col] /
                    ↪ temp[row][col];
                for (int j=col; j<m; ++j)
                    temp[i][j] -= temp[row][j] * c;
            }
        ++row;
    }
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            inverse[i][j] = temp[i][j+n];
    return true;
}

```

1.4.8. Adjoint Matrix

```

Matrix<T> Adjunta(){
    int n = A[0].size();
    Matrix<int> adjoint(n);
    Matrix<double> inverse(n);
    this->Inverse(inverse);
    int determinante = this->detGauss();
    if(determinante){
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                adjoint[i][j] =
                    ↪ (T)round((inverse[i][j]*determinante));
    }
    else {
        adjoint = this->cofactorMatrix().transpose();
    }
    return adjoint;
}

```

1.4.9. Recurrencias lineales

```

lli Linear_recurrence(vector<lli> C, vector<lli> init, lli
    ↪ n, bool constante){
    int k = C.size();

```

```

Matrix<lli> T(k,k);
Matrix<lli> first(k,1);
for(int i = 0;i<k;i++)T[0][i] = C[i];
for(int i = 0,col=1;i<k && col<k;i++,col++)
    T[col][i]=1;
if(constante){
    for(int i = 0;i<k;i++)first[i][0]=init[(k-2)-i];
    first[k-1][0]=init[k-1];
}
else
    for(int i = 0;i<k;i++)first[i][0]=init[(k-1)-i];
if(constante)
    T^=((n-k)+1);
else
    T^=(n-k);
Matrix<lli> sol = T*first;
return sol[0][0];
}

```

1.4.10. Kirchhoff Matrix Tree Theorem

Count the number of spanning trees in a graph, as the determinant of the Laplacian matrix of the graph.

Laplacian Matrix :

Given a simple graph G with n vertices, its Laplacian matrix $L_{n \times n}$ is defined as

$$L = D - A$$

The elements of L are given by

$$L_{i,j} = \begin{cases} \deg(v_i) & \text{if } i == j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

define $\tau(G)$ as number of spanning trees of a grap G

$$\tau(G) = \det L_{n-1 \times n-1}$$

Where $L_{n-1 \times n-1}$ is a laplacian matrix deleting any row and any column

$$\det \begin{pmatrix} \deg(v_1) & L_{1,2} & \cdots & L_{1,n-1} \\ L_{2,1} & \deg(v_2) & \cdots & L_{2,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ L_{n-1,1} & L_{n-1,2} & \cdots & \deg(v_{n-1}) \end{pmatrix}$$

Generalization for a multigraph $K_n^m \pm G$

define $\tau(K_n^m \pm G)$ as number of spanning trees of a grap $K_n^m \pm G$

$$\tau(K_n^m \pm G) = n * (nm)^{n-p-2} \det(B)$$

where $B = mnI_p + \alpha * L(G)$ is a $p \times p$ matrix, $\alpha = \pm$ according $(K_n^m \pm G)$, and $L(G)$ is the Kirchhoff matrix of G

```

cin>>n>>m>>k;
Matrix<lli> Kirchof(n);
for(int i = 0;i<m;i++){
    cin>>a>>b;
    a--;
    b--;
    Kirchof[a][b] = Kirchof[b][a] = 1;
    Kirchof[a][a]++;
    Kirchof[b][b]++;
}
for(int i =0;i<n;i++)
    Kirchof[i][i] =
        (1ll*n*k%mod-Kirchof[i][i]+mod)%mod;
lli ans = 1;
ans = ans*(mod_pow(1ll*k*n%mod*k%mod*n%mod,mod-2));
lli determinante =Kirchof.det();
ans = ans*(mod_pow(determinante,k))%mod;
cout<<ans<<endl;

```

1.5. Combinatorics

1.5.1. Binomial coefficients

```

/*****
    Binomial coefficients
    * Computes C(n,k)
    * Tested [?]
*****/
O(n) solutions
    Based in DP
    Based in the prof of  $C(n,k) = C(n-1,k-1) + C(n-1,k)$ 
    Also calc all  $C(n,i)$  for  $0 \leq i \leq n$ 
*/
long binomial_Coeff(int n, int m){
    int i,j;
    long bc[MAXN][MAXN];
    for (i=0; i<=n; i++) bc[i][0] = 1;
    for (j=0; j<=n; j++) bc[j][j] = 1;
    for (i=1; i<=n; i++)
        for (j=1; j<i; j++)
            bc[i][j] = bc[i-1][j-1] + bc[i-1][j];
    return bc[n][m];
}

/*
    O(k) solution
    Only calc C(n,k)
*/
int binomial_Coeff_2(int n, int k) {
    int res = 1;
    if ( k > n - k )
        k = n - k;
    for (int i = 0; i < k; ++i){
        res *= (n - i);
        res /= (i + 1);
    }
    return res;
}

/*
    O(k) solution
    Only calc C(n,k)
*/

```

```

int binomial_Coeff_3(int n, int k){
    vector<int> C(k+1,0);
    C[0] = 1; // nC0 is 1
    for (int i = 1; i <= n; i++) {
        for (int j = min(i, k); j > 0; j--)
            C[j] = C[j] + C[j-1];
    }
    return C[k];
}

/*****
    * Factorial modulo P
    If only need one factorial
    O(P logp n)
    * Tested [?]
*****/
int factmod(int n, int p) {
    int res = 1;
    while (n > 1) {
        res = (res * ((n/p) % 2 ? p-1 : 1)) % p;
        for (int i = 2; i <= n%p; ++i)
            res = (res * i) % p;
        n /= p;
    }
    return res % p;
}

/*****
    Lucas Theorem
    * Computes C(N,R)%p in O(log(n)) if P is prime
    * Tested [Codeforces D - Sasha and Interesting Fact from
    ↳ Graph Theory]
*****/
/*
    Precalc
    -Inverse modular to n
    -Factorial modulo p
    -Inverse modular of factorial
*/
const int M = 1e6;
const lli mod = 986444681;

```

```

vector<lli> fact(M+1, 1), inv(M+1, 1), invfact(M+1, 1);
lli ncr(lli n, lli r){
    if(r < 0 || r > n) return 0;
    return fact[n] * invfact[r] % mod * invfact[n - r] % mod;
}

void calc(int m){
    for(int i = 2; i ≤ M; ++i){
        fact[i] = (lli)fact[i-1] * i % mod;
        inv[i] = mod - (lli)inv[mod % i] * (mod / i) % mod;
        invfact[i] = (lli)invfact[i-1] * inv[i] % mod;
    }
}

/*
    Lucas Theorem
*/
lli Lucas(lli N, lli R){
    if(R < 0 || R > N)
        return 0;
    if(R = 0 || R = N)
        return 1ll;
    if(N ≥ mod)
        return (1ll*Lucas(N/mod, R/mod)*Lucas(N%mod, R%mod))%mod;
    return fact[n] * invfact[r] % mod * invfact[n - r] % mod;
}

/*
    Using calc() we can also calculate P(n,k)
    ↪ (permutations)
*/
lli permutation(int n, int k){
    return (1ll*fact[n] * invfact[n-k])%mod;
}

//    Computes C(N,R)%p
lli power(lli x, lli y, lli p) {
    lli res = 1;
    x = x % p;
    while (y > 0) {
        if (y & 1)
            res = (res*x) % p;
        y = y>>1;

```

```

        x = (x*x) % p;
    }
    return res;
}

lli modInverse(lli n, lli p) {
    return power(n, p-2, p);
}

lli nCrModPFermat(lli n, lli r, lli p) {
    if (r==0)
        return 1;
    lli fac[n+1];
    fac[0] = 1;
    for (lli i=1 ; i≤n; i++)
        fac[i] = fac[i-1]*i%p;
    return (fac[n]* modInverse(fac[r], p) % p *
            modInverse(fac[n-r], p) % p) % p;
}

/*****
 * Cayley's formula
 * Computes all posibles trees whit n nodes
 * Tested [Codeforces D - Sasha and Interesting Fact from
 * ↪ Graph Theory]
 *****/
lli cayley(int n ,int k){
    if(n-k-1<0)
        return (1ll*k*mod_pow(n,mod-2))%mod;
    return (1ll*k*mod_pow(n,n-k-1))%mod;
}

```

Distribute N items in m container $\binom{N+m-1}{N}$

2. Strings

2.1. Trie

```

struct Trie{
    unordered_map<char, Trie*> child;
    int prefix;
    bool end;
};

struct Trie *getNode(){
    struct Trie *p = new Trie;
    return p;
}

void insert(struct Trie *root, string key){
    struct Trie *S = root;
    for(int i = 0 ; i < key.length(); i++){
        if(S->child.find(key[i]) == S->child.end()){
            S->child[key[i]] = getNode();

            S = S->child[key[i]];
            S->prefix++;
        }
        S->end = true;
    }

    bool search(struct Trie *root, string key){
        struct Trie *S = root;
        int n = key.size();
        for(int i = 0; i < n; i++){
            if(S->child.find(key[i]) == S->child.end()){
                return false;
            }
            S = S->child[key[i]];
        }
        if(S->end) return true;
        else
            return false;
    }

    int countprefixes(Trie* root, string s){
        int n = s.size();

```

```

        Trie* mov = root;
        for(int i=0; i<n; i++){
            if(mov->child.find(s[i]) == mov->child.end())
                return 0;
            mov = mov->child[s[i]];
        }
        return mov->prefix;
    }

    Trie* remove(Trie* root, string word, int depth = 0){
        if(!root)
            return NULL;
        if (depth == word.size()) {
            if (root->end)
                root->end = false;
            if (root->child.size()) {
                delete (root);
                root = NULL;
            }
            return root;
        }
        root->child[word[depth]] =
            remove(root->child[word[depth]], word, depth + 1);
        if (root->child.size() == 0 && root->end == false) {
            delete(root);
            root = NULL;
        }
        return root;
    }

    void print(Trie* root, char str[], int level){
        if(root->end){
            str[level] = '\\0';
            cout<<str<<endl;
        }
        for(auto c:root->child){
            str[level] = c.first;
            print(c.second, str, level+1);
        }
    }
}

```


2.2. Suffix array and LCP

```

void radix_sort(vector<int> &P,vector<int> &c){
    int n = P.size();
    vector<int> cnt(n);
    for(auto d:c)
        cnt[d]++;
    vector<int> pos(n);
    vector<int> nP(n);
    pos[0]= 0;
    for(int i = 1;i<n;i++){
        pos[i] = pos[i-1]+cnt[i-1];
    }
    for(auto d:P){
        int i = c[d];
        nP[pos[i]] = d;
        pos[i]++;
    }
    P = nP;
}

int main(){__
    string s;
    cin>>s;
    s+='#';
    int n = s.size();
    vector<int>c(n);
    vector<int>p(n);
    vector<pair<char,int>> a(n);
    for(int i = 0;i<n;i++)a[i] = {s[i],i};
    sort(a.begin(),a.end());
    for(int i = 0;i<n;i++){
        p[i] = a[i].second;
        c[p[0]] = 0;
        for(int i = 1;i<n;i++){
            if(a[i].first == a[i-1].first)
                c[p[i]] = c[p[i-1]];
            else c[p[i]] = c[p[i-1]]+1;
        }
    }

    int k = 0;
    while((1<<k)<n){
        for(int i = 0 ;i<n;i++)

```

```

        p[i] = ((p[i]-(1<<k))+n)%n;
        radix_sort(p,c);
        vector<int> nC(n);
        nC[p[0]] = 0;
        for(int i = 1;i<n;i++){
            pair<int,int> prev = {c[p[i-1]],c[(p[i-1]+
            ↪ (1<<k))%n]};
            pair<int,int> now = {c[p[i]],c[(p[i]+
            ↪ (1<<k))%n]};
            if(prev == now)
                nC[p[i]] = nC[p[i-1]];
            else nC[p[i]] = nC[p[i-1]]+1;
        }
        c = nC;
        k++;
    }

    // LCP O(n)
    k = 0;
    vector<int> lcp(n);
    for(int i = 0;i<n-1;i++){
        int x = c[i];
        int j = p[x-1];
        while(s[i+k] == s[j+k])k++;
        lcp[x] = k;
        k = max(k-1,0);
    }

    for(int i = 0;i<n;i++)cout<<lcp[i]<<" " <<p[i]<<"
    ↪ "<<s.substr(p[i],n-p[i])<<endl;
    cout<<endl;
    return 0;
}

```

2.3. Suffix Tree

```

const int inf = 1e9;
const int maxn = 1e6 ;
char s[maxn];
map<int, int> to[maxn];
int len[maxn], start[maxn], link[maxn];

```

```

int node, remaind;
int sz = 1, n = 0;
int make_node(int _pos, int _len){
    start[sz] = _pos;
    len[sz] = _len;
    return sz++;
}

void go_edge(){
    while(remaind > len[to[node][s[n - remaind]]]){
        // cout<<"MAYOR"<<endl;
        node = to[node][s[n - remaind]];
        remaind -= len[node];
    }
}

void add_letter(int c){
    s[n++] = c;
    remaind++;
    // cout<<"suffix " <<remaind<<" " <<s[n-1]<<endl;
    int last = 0;
    while(remaind > 0){
        go_edge();
        int edge = s[n - remaind];
        int &v = to[node][edge];
        // cout<<v<<" " <<start[v]<<" " <<(char)edge<<"
        //   <<"<<remaind<<endl;
        int t = s[start[v] + remaind - 1];
        // cout<<(char)t<<endl;
        if(v == 0){
            // cout<<"if"<<endl;
            v = make_node(n - remaind, inf);
            // cout<<" "<<v<<endl;
            link[last] = node;
            last = 0;
        }
        else if(t == c){
            // cout<<"else if"<<endl;
            link[last] = node;
            return;
        }
        else{

```

```

        // cout<<"else"<<endl;
        int u = make_node(start[v], remaind - 1);
        to[u][c] = make_node(n - 1, inf);
        to[u][t] = v;
        start[v] += remaind - 1;
        len[v] -= remaind - 1;
        // cout<<len[v]<<endl;
        v = u;
        link[last] = u;
        last = u;
    }
    if(node == 0)
        remaind--;
    else
        node = link[node];
}

bool dfsForPrint(int node, char edge){
    if(node != 0)
        // cout<<edge<<" " <<node<<" " <<len[node]<<"
        //   <<"<<start[node]<<endl;
        for(auto c:to[node])
            dfsForPrint(c.second, c.first);

    return 0;
}

int main(){
    clock_t begin = clock();
    len[0] = inf;
    string s = "abcabxabcd";
    int ans = 0;
    for(int i = 0; i < s.size(); i++)
        add_letter(s[i]);
    clock_t end = clock();
    double time_spent = (double)(end - begin) /
        //   <<CLOCKS_PER_SEC;
    cout<<fixed<<setprecision(15)<<time_spent<<endl;
    // for(int i = 1; i < sz; i++)
    //     ans += min((int)s.size() - start[i], len[i]);
    // cout << ans << "\n";
    return 0;
}

```

2.4. Suffix automaton

2.5. Aho corasick

2.6. Z function

2.7. Knuth morris pratt

```
vector<int> p_function(const string& v){
    int n = v.size();
    vector<int> p(n);
    for(int i = 1; i < n; i++){
        int j = p[i - 1];
        while(j > 0 && v[j] != v[i]){
            j = p[j - 1];
        }
        if(v[j] == v[i])
            j++;
        p[i] = j;
    }
    return p;
}
```

2.8. Palindromic tree

2.9. Manacher

```
using namespace std;
int main(){
    string s;
    cin>>s;
    int n = s.size();
    vector<vector<int>> p(2,vector<int>(n,0));
    for(int z=1,l=0,r=0;z<2;z++,l=0,r=0)
        for(int i=0;i<n;i++)
        {
            if(i<r) p[z][i]=min(r-i+!z,p[z][l+r-i+!z]);
            int L=i-p[z][i], R=i+p[z][i]-!z;
            cout<<L<<" "<<R<<" "<<(L-1>=0) <<"
                <<"<<(R+1<n)<<endl;
            while(L-1>=0 && R+1<n && s[L-1]==s[R+1])
                <<"<<p[z][i]++,L--,R++;
            if(R>r) l=L,r=R;
        }
}
```

```
    }
    for(int i = 0;i<n;i++)cout<<p[0][i]<<"
        <<"<<p[1][i]<<endl;
}
```

2.10. Rolling hashes

```
// Generate random base in (before, after) open interval:
int gen_base(const int before, const int after) {
    auto seed =
        <<"<< chrono::high_resolution_clock::now().time_since_epoch().count();
    seed ^= ull(new ull);
    mt19937 mt_rand(seed);
    int base = uniform_int_distribution<int>(before+1,
        <<"<< after)(mt_rand);
    return base % 2 == 0 ? base-1 : base;
}
```

```
struct PolyHash {
    // ----- Static variables -----
    static vector<int> pow1;          // powers of base
    <<"<< modulo mod
    static vector<ull> pow2;          // powers of base
    <<"<< modulo 2^64
    static int base;                  // base (point of
    <<"<< hashing)

    // ----- Static functions -----
    static inline int diff(int a, int b) {
        // Diff between `a` and `b` modulo mod (0 ≤ a < mod,
        <<"<< 0 ≤ b < mod)
        return (a - b) < 0 ? a + 2147483647 : a;
    }

    static inline int mod(ull x) {
        x += 2147483647;
        x = (x >> 31) + (x & 2147483647);
        return int((x >> 31) + (x & 2147483647));
    }
}
```

```
// ----- Variables of class -----
vector<int> pref1; // Hash on prefix modulo mod
vector<ull> pref2; // Hash on prefix modulo 2^64

// Get power of base by modulo mod:
inline int get_pow1(int p) const {
    static int __base[4] = {1, base, mod(ull(base) *
    ↪ base), mod(mod(ull(base) * base) * ull(base))};
    return mod(ull(__base[p % 4]) * pow1[p / 4]);
}

// Get power of base by modulo 2^64:
inline ull get_pow2(int p) const {
    static ull __base[4] = {ull(1), ull(base),
    ↪ ull(base) * base, ull(base) * base * base};
    return pow2[p / 4] * __base[p % 4];
}

// Constructor from string:
PolyHash(const string& s)
    : pref1(s.size()+1u, 0)
    , pref2(s.size()+1u, 0)
{
    const int n = s.size();
    pow1.reserve((n+3)/4);
    pow2.reserve((n+3)/4);
    // Firstly calculated needed power of base:
    int pow1_4 = mod(ull(base) * base); // base^2 mod
    ↪ 2^31-1
    pow1_4 = mod(ull(pow1_4) * pow1_4); // base^4 mod
    ↪ 2^31-1
    ull pow2_4 = ull(base) * base; // base^2 mod
    ↪ 2^64
    pow2_4 *= pow2_4; // base^4 mod
    ↪ 2^64
    while (4 * (int)pow1.size() ≤ n) {
        pow1.push_back(mod((ull)pow1.back() * pow1_4));
        pow2.push_back(pow2.back() * pow2_4);
    }
    int curr_pow1 = 1;
    ull curr_pow2 = 1;

```

```
for (int i = 0; i < n; ++i) { // Fill arrays with
    ↪ polynomial hashes on prefix
    assert(base > s[i]);
    pref1[i+1] = mod(pref1[i] + (ull)s[i] *
    ↪ curr_pow1);
    pref2[i+1] = pref2[i] + s[i] * curr_pow2;
    curr_pow1 = mod((ull)curr_pow1 * base);
    curr_pow2 *= base;
}

// Polynomial hash of subsequence [pos, pos+len)
// If mxPow ≠ 0, value automatically multiply on base
↪ in needed power. Finally base ^ mxPow
inline pair<int, ull> operator()(const int pos, const
    ↪ int len, const int mxPow = 0) const {
    int hash1 = pref1[pos+len] - pref1[pos];
    ull hash2 = pref2[pos+len] - pref2[pos];
    if (hash1 < 0) hash1 += 2147483647;
    if (mxPow ≠ 0) {
        hash1 = mod((ull)hash1 *
        ↪ get_pow1(mxPow-(pos+len-1)));
        hash2 *= get_pow2(mxPow-(pos+len-1));
    }
    return make_pair(hash1, hash2);
}

};

// Init static variables of PolyHash class:
int PolyHash::base((int)1e9+7);
vector<int> PolyHash::pow1{1};
vector<ull> PolyHash::pow2{1};

int main() {
    string a;
    {
        vector<char> buf(1+1000000);
        scanf("%1000000s", &buf[0]);
        a = string(&buf[0]);
    }

    // Gen random base of hashing:

```

```

PolyHash::base = gen_base(256, 2147483647);

// Construct polynomial hashes on prefix of original
//   ↪ and reversed string:
PolyHash hash_a(a);
reverse(a.begin(), a.end());
PolyHash hash_b(a);

// Get length of strings (mxPow == n)
const int n = (int)a.size();

ull answ = 0;
for (int i = 0, j = n-1; i < n; ++i, --j) {
    // Palindromes odd length:
    int low = 0, high = min(n-i, n-j)+1;
    while (high - low > 1) {
        int mid = (low + high) / 2;
        if (hash_a(i, mid, n) == hash_b(j, mid, n)) {
            low = mid;
        } else {
            high = mid;
        }
    }
    answ += low;
    // Palindromes even length:
    low = 0, high = min(n-i-1, n-j)+1;
    while (high - low > 1) {
        int mid = (low + high) / 2;
        if (hash_a(i+1, mid, n) == hash_b(j, mid, n)) {
            low = mid;
        } else {
            high = mid;
        }
    }
    answ += low;
}
cout << answ;
return 0;
}

```