Grimoire's Standard Code Library*

Shanghai Jiao Tong University

Dated: 2017 年 10 月 15 日

*https://github.com/kzoacn/Grimoire

目录

Chapter 1

代数

$O(n^2 \log n)$ 求线性递推数列第 n 项

Given $a_0, a_1, \cdots, a_{m-1}$
$a_n = c_0 * a_{n-m} + \cdots + c_{m-1} * a_0$
a_0 *is the nth element*, $\cdots$, a_{m-1} *is the* $n+m-1$*th element*

```
void linear_recurrence(long long n, int m, int a[], int c[], int p) {
    long long v[M] = {1 % p}, u[M << 1], msk = !!n;
    for(long long i(n); i > 1; i >>= 1) {
        msk <<= 1;
    }
    for(long long x(0); msk; msk >>= 1, x <<= 1) {
        fill_n(u, m << 1, 0);
        int b(!!(n & msk));
        x |= b;
        if(x < m) {
            u[x] = 1 % p;
        }else {
            for(int i(0); i < m; i++) {
                for(int j(0), t(i + b); j < m; j++, t++) {
                    u[t] = (u[t] + v[i] * v[j]) % p;
                }
            }
            for(int i((m << 1) - 1); i >= m; i--) {
                for(int j(0), t(i - m); j < m; j++, t++) {
                    u[t] = (u[t] + c[j] * u[i]) % p;
                }
            }
        }
        copy(u, u + m, v);
    }
    //a[n] = v[0] * a[0] + v[1] * a[1] + ... + v[m - 1] * a[m - 1].
    for(int i(m); i < 2 * m; i++) {
        a[i] = 0;
```

```
        for(int j(0); j < m; j++) {
            a[i] = (a[i] + (long long)c[j] * a[i + j - m]) % p;
        }
    }
    for(int j(0); j < m; j++) {
        b[j] = 0;
        for(int i(0); i < m; i++) {
            b[j] = (b[j] + v[i] * a[i + j]) % p;
        }
    }
    for(int j(0); j < m; j++) {
        a[j] = b[j];
    }
}
```

任意模数快速傅里叶变换

```
// double 精度对 10^9 + 7 取模最多可以做到 2^20
const int MOD = 1000003;
const double PI = acos(-1);
typedef complex<double> Complex;
const int N = 65536, L = 15, MASK = (1 << L) - 1;
Complex w[N];
void FFTInit() {
    for (int i = 0; i < N; ++i)
        w[i] = Complex(cos(2 * i * PI / N), sin(2 * i * PI / N));
}
void FFT(Complex p[], int n) {
    for (int i = 1, j = 0; i < n - 1; ++i) {
        for (int s = n; j ^= s >>= 1, ~j & s;);
        if (i < j) swap(p[i], p[j]);
    }
    for (int d = 0; (1 << d) < n; ++d) {
        int m = 1 << d, m2 = m * 2, rm = n >> (d + 1);
        for (int i = 0; i < n; i += m2) {
            for (int j = 0; j < m; ++j) {
                Complex &p1 = p[i + j + m], &p2 = p[i + j];
                Complex t = w[rm * j] * p1;
                p1 = p2 - t, p2 = p2 + t;
            } } }
}
Complex A[N], B[N], C[N], D[N];
void mul(int a[N], int b[N]) {
    for (int i = 0; i < N; ++i) {
        A[i] = Complex(a[i] >> L, a[i] & MASK);
```

```
        B[i] = Complex(b[i] >> L, b[i] & MASK);
    }
    FFT(A, N), FFT(B, N);
    for (int i = 0; i < N; ++i) {
        int j = (N - i) % N;
        Complex da = (A[i] - conj(A[j])) * Complex(0, -0.5),
                db = (A[i] + conj(A[j])) * Complex(0.5, 0),
                dc = (B[i] - conj(B[j])) * Complex(0, -0.5),
                dd = (B[i] + conj(B[j])) * Complex(0.5, 0);
        C[j] = da * dd + da * dc * Complex(0, 1);
        D[j] = db * dd + db * dc * Complex(0, 1);
    }
    FFT(C, N), FFT(D, N);
    for (int i = 0; i < N; ++i) {
        long long da = (long long)(C[i].imag() / N + 0.5) % MOD,
                  db = (long long)(C[i].real() / N + 0.5) % MOD,
                  dc = (long long)(D[i].imag() / N + 0.5) % MOD,
                  dd = (long long)(D[i].real() / N + 0.5) % MOD;
        a[i] = ((dd << (L * 2)) + ((db + dc) << L) + da) % MOD;
    }
}
```

快速傅里叶变换

```
int prepare(int n) {
    int len = 1;
    for (; len <= 2 * n; len <<= 1);
    for (int i = 0; i < len; i++) {
        e[0][i] = Complex(cos(2 * pi * i / len), sin(2 * pi * i / len));
        e[1][i] = Complex(cos(2 * pi * i / len), -sin(2 * pi * i / len));
    }
    return len;
}
void DFT(Complex *a, int n, int f) {
    for (int i = 0, j = 0; i < n; i++) {
        if (i > j) std::swap(a[i], a[j]);
        for (int t = n >> 1; (j ^= t) < t; t >>= 1);
    }
    for (int i = 2; i <= n; i <<= 1)
        for (int j = 0; j < n; j += i)
            for (int k = 0; k < (i >> 1); k++) {
                Complex A = a[j + k];
                Complex B = e[f][n / i * k] * a[j + k + (i >> 1)];
                a[j + k] = A + B;
                a[j + k + (i >> 1)] = A - B;
```

```
            }
    if (f == 1) {
        for (int i = 0; i < n; i++)
            a[i].a /= n;
    }
}
```

闪电数论变换与魔力 CRT

```
#define meminit(A, l, r) memset(A + (l), 0, sizeof(*A) * ((r) - (l)))
#define memcopy(B, A, l, r) memcpy(B, A + (l), sizeof(*A) * ((r) - (l)))
void DFT(int *a, int n, int f) { //f=1 逆 DFT
    for (register int i = 0, j = 0; i < n; i++) {
        if (i > j) std::swap(a[i], a[j]);
        for (register int t = n >> 1; (j ^= t) < t; t >>= 1);
    }
    for (register int i = 2; i <= n; i <<= 1) {
        static int exp[MAXN];
        exp[0] = 1; exp[1] = fpm(PRT, (MOD - 1) / i, MOD);
        if (f == 1) exp[1] = fpm(exp[1], MOD - 2, MOD);
        for (register int k = 2; k < (i >> 1); k++) {
            exp[k] = 1ll * exp[k - 1] * exp[1] % MOD;
        }
        for (register int j = 0; j < n; j += i) {
            for (register int k = 0; k < (i >> 1); k++) {
                register int &pA = a[j + k], &pB = a[j + k + (i >> 1)];
                register long long B = 1ll * pB * exp[k];
                pB = (pA - B) % MOD;
                pA = (pA + B) % MOD;
            }
        }
    }
    if (f == 1) {
        register int rev = fpm(n, MOD - 2, MOD);
        for (register int i = 0; i < n; i++) {
            a[i] = 1ll * a[i] * rev % MOD;
            if (a[i] < 0) { a[i] += MOD; }
        }
    }
}
// 在不写高精度的情况下合并 FFT 所得结果对 MOD 取模后的答案
// 值得注意的是，这个东西不能最后再合并，而是应该每做一次多项式乘法就 CRT 一次
int CRT(int *a) {
    static int x[3];
    for (int i = 0; i < 3; i++) {
```

```
        x[i] = a[i];
        for (int j = 0; j < i; j++) {
            int t = (x[i] - x[j] + FFT[i] -> MOD) % FFT[i] -> MOD;
            if (t < 0) t += FFT[i] -> MOD;
            x[i] = 1LL * t * inv[j][i] % FFT[i] -> MOD;
        }
    }
    int sum = 1, ret = x[0] % MOD;
    for (int i = 1; i < 3; i ++) {
        sum = 1LL * sum * FFT[i - 1] -> MOD % MOD;
        ret += 1LL * x[i] * sum % MOD;
        if(ret >= MOD) ret -= MOD;
    }
    return ret;
}
for (int i = 0; i < 3; i++) // inv 数组的预处理过程，inverse(x, p) 表示求 x 在 p 下逆元
    for (int j = 0; j < 3; j++)
        inv[i][j] = inverse(FFT[i] -> MOD, FFT[j] -> MOD);
```

多项式求逆

Given polynomial a and n, b is the polynomial such that $a * b \equiv 1 \pmod{x^n}$

```
void getInv(int *a, int *b, int n) {
    static int tmp[MAXN];
    b[0] = fpm(a[0], MOD - 2, MOD);
    for (int c = 2, M = 1; c < (n << 1); c <<= 1) {
        for (; M <= 3 * (c - 1); M <<= 1);
        meminit(b, c, M);
        meminit(tmp, c, M);
        memcopy(tmp, a, 0, c);
        DFT(tmp, M, 0);
        DFT(b, M, 0);
        for (int i = 0; i < M; i++) {
            b[i] = 1ll * b[i] * (2ll - 1ll * tmp[i] * b[i] % MOD + MOD) % MOD;
        }
        DFT(b, M, 1);
        meminit(b, c, M);
    }
}
```

多项式除法

d is quotient and r is remainder

```
void divide(int n, int m, int *a, int *b, int *d, int *r) { // n、m 分别为多项式 A（被除数）
  ↪ 和 B（除数）的指数 + 1
    static int M, tA[MAXN], tB[MAXN], inv[MAXN], tD[MAXN];
    for (; n > 0 && a[n - 1] == 0; n--);
    for (; m > 0 && b[m - 1] == 0; m--);
    for (int i = 0; i < n; i++) tA[i] = a[n - i - 1];
    for (int i = 0; i < m; i++) tB[i] = b[m - i - 1];
    for (M = 1; M <= n - m + 1; M <<= 1);
    if (m < M) meminit(tB, m, M);
    getInv(tB, inv, M);
    for (M = 1; M <= 2 * (n - m + 1); M <<= 1);
    meminit(inv, n - m + 1, M);
    meminit(tA, n - m + 1, M);
    DFT(inv, M, 0);
    DFT(tA, M, 0);
    for (int i = 0; i < M; i++) {
        d[i] = 1ll * inv[i] * tA[i] % MOD;
    }
    DFT(d, M, 1);
    std::reverse(d, d + n - m + 1);
    for (M = 1; M <= n; M <<= 1);
    memcopy(tB, b, 0, m);
    if (m < M) meminit(tB, m, M);
    memcopy(tD, d, 0, n - m + 1);
    meminit(tD, n - m + 1, M);
    DFT(tD, M, 0);
    DFT(tB, M, 0);
    for (int i = 0; i < M; i++) {
        r[i] = 1ll * tD[i] * tB[i] % MOD;
    }
    DFT(r, M, 1);
    meminit(r, n, M);
    for (int i = 0; i < n; i++) {
        r[i] = (a[i] - r[i] + MOD) % MOD;
    }
}
```

多项式取指数取对数

Given polynomial a and n, b is the polynomial such that $b \equiv e^a \pmod{x^n}$ or $b \equiv \ln a \pmod{x^n}$

```
void getDiff(int *a, int *b, int n) { // 多项式取微分
    for (int i = 0; i + 1 < n; i++) {
        b[i] = 1ll * (i + 1) * a[i + 1] % MOD;
    }
```

```cpp
    b[n - 1] = 0;
}
void getInt(int *a, int *b, int n) { // 多项式取积分，积分常数为 0
    static int inv[MAXN];
    inv[1] = 1;
    for (int i = 2; i < n; i++) {
        inv[i] = 1ll * (MOD - MOD / i) * inv[MOD % i] % MOD;
    }
    b[0] = 0;
    for (int i = 1; i < n; i++) {
        b[i] = 1ll * a[i - 1] * inv[i] % MOD;
    }
}
void getLn(int *a, int *b, int n) {
    static int inv[MAXN], d[MAXN];
    int M = 1;
    for (; M <= 2 * (n - 1); M <<= 1);
    getInv(a, inv, n);
    getDiff(a, d, n);
    meminit(d, n, M);
    meminit(inv, n, M);
    DFT(d, M, 0); DFT(inv, M, 0);
    for (int i = 0; i < M; i++) {
        d[i] = 1ll * d[i] * inv[i] % MOD;
    }
    DFT(d, M, 1);
    getInt(d, b, n);
}
void getExp(int *a, int *b, int n) {
    static int ln[MAXN], tmp[MAXN];
    b[0] = 1;
    for (int c = 2, M = 1; c < (n << 1); c <<= 1) {
        for (; M <= 2 * (c - 1); M <<= 1);
        int bound = std::min(c, n);
        memcopy(tmp, a, 0, bound);
        meminit(tmp, bound, M);
        meminit(b, c, M);
        getLn(b, ln, c);
        meminit(ln, c, M);
        DFT(b, M, 0);
        DFT(tmp, M, 0);
        DFT(ln, M, 0);
        for (int i = 0; i < M; i++) {
            b[i] = 1ll * b[i] * (1ll - ln[i] + tmp[i] + MOD) % MOD;
        }
        DFT(b, M, 1);
```

```
        meminit(b, c, M);
    }
}
```

快速沃尔什变换

```
void FWT(LL a[],int n,int ty){
    for(int d=1;d<n;d<<=1){
        for(int m=(d<<1),i=0;i<n;i+=m){
            if(ty==1){
                for(int j=0;j<d;j++){
                    LL x=a[i+j],y=a[i+j+d];
                    a[i+j]=x+y;
                    a[i+j+d]=x-y;
                    //xor:a[i+j]=x+y,a[i+j+d]=x-y;
                    //and:a[i+j]=x+y;
                    //or:a[i+j+d]=x+y;
                }
            }else{
                for(int j=0;j<d;j++){
                    LL x=a[i+j],y=a[i+j+d];
                    a[i+j]=(x+y)/2;
                    a[i+j+d]=(x-y)/2;
                    //xor:a[i+j]=(x+y)/2,a[i+j+d]=(x-y)/2;
                    //and:a[i+j]=x-y;
                    //or:a[i+j+d]=y-x;
                }
            }
        }
    }
}
    FWT(a,1<<n,1);
    FWT(b,1<<n,1);
    for(int i=0;i<(1<<n);i++)
        c[i]=a[i]*b[i];
    FWT(c,1<<n,-1);
```

Chapter 2

数论

大整数相乘取模

```
// x 与 y 须非负
long long mult(long long x, long long y, long long MODN) {
    long long t = (x * y - (long long)((long double)x / MODN * y + 1e-3) * MODN) % MODN;
    return t < 0 ? t + MODN : t;
}
```

EX-GCD

```
LL exgcd(LL a,LL b,LL &x,LL &y){
    if(!b){
        x=1;y=0;return a;
    }else{
        LL d=exgcd(b,a%b,x,y);
        LL t=x;x=y;y=t-a/b*y;
        return d;
    }
}
```

Miller-rabin

```
const int BASE[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
bool check(long long n,int base) {
    long long n2=n-1,res;
    int s=0;
    while(n2%2==0) n2>>=1,s++;
    res=pw(base,n2,n);
    if((res==1)||(res==n-1)) return 1;
    while(s--) {
        res=mul(res,res,n);
```

```cpp
        if(res==n-1) return 1;
    }
    return 0; // n is not a strong pseudo prime
}
bool isprime(const long long &n) {
    if(n==2)
        return true;
    if(n<2 || n%2==0)
        return false;
    for(int i=0;i<12&&BASE[i]<n;i++){
        if(!check(n,BASE[i]))
            return false;
    }
    return true;
}
```

Pollard-rho.cpp

```cpp
LL prho(LL n,LL c){
    LL i=1,k=2,x=rand()%(n-1)+1,y=x;
    while(1){
        i++;x=(x*x%n+c)%n;
        LL d=__gcd((y-x+n)%n,n);
        if(d>1&&d<n)return d;
        if(y==x)return n;
        if(i==k)y=x,k<<=1;
    }
}
void factor(LL n,vector<LL>&fat){
    if(n==1)return;
    if(isprime(n)){
        fat.push_back(n);
        return;
    }LL p=n;
    while(p>=n)p=prho(p,rand()%(n-1)+1);
    factor(p,fat);
    factor(n/p,fat);
}
```

非互质 CRT

first is remainder, second is module

```cpp
inline void fix(LL &x, LL y) {
    x = (x % y + y) % y;
```

```
}
bool solve(int n, std::pair<LL, LL> a[],
                  std::pair<LL, LL> &ans) {
    ans = std::make_pair(1, 1);
    for (int i = 0; i < n; ++i) {
        LL num, y;
        euclid(ans.second, a[i].second, num, y);
        LL divisor = std::__gcd(ans.second, a[i].second);
        if ((a[i].first - ans.first) % divisor) {
            return false;
        }
        num *= (a[i].first - ans.first) / divisor;
        fix(num, a[i].second);
        ans.first += ans.second * num;
        ans.second *= a[i].second / divisor;
        fix(ans.first, ans.second);
    }
    return true;
}
```

非互质 CRT -zky

```
//merge Ax=B and ax=b to A'x=B'
LL china(int n,int *a,int *m){
    LL M=1,d,x=0,y;
    for(int i=0;i<n;i++)
        M*=m[i];
    for(int i=0;i<n;i++){
        LL w=M/m[i];
        d=exgcd(m[i],w,d,y);
        y=(y%M+M)%M;
        x=(x+y*w%M*a[i])%M;
    }
    while(x<0)x+=M;
    return x;
}
void merge(LL &A,LL &B,LL a,LL b){
    LL x,y;
    sol(A,-a,b-B,x,y);
    A=lcm(A,a);
    B=(a*y+b)%A;
    B=(B+A)%A;
}
```

Pell 方程

```
// $x_{k+1} = x_0 x_k + n y_0 y_k$
// $y_{k+1} = x_0 y_k + y_0 x_k$
// n is not the index of which you want
pair<ll, ll> pell(ll n) {
    static ll p[N], q[N], g[N], h[N], a[N];
    p[1] = q[0] = h[1] = 1; p[0] = q[1] = g[1] = 0;
    a[2] = (ll)(floor(sqrtl(n) + 1e-7L));
    for(int i = 2; ; i ++) {
        g[i] = -g[i - 1] + a[i] * h[i - 1];
        h[i] = (n - g[i] * g[i]) / h[i - 1];
        a[i + 1] = (g[i] + a[2]) / h[i];
        p[i] = a[i] * p[i - 1] + p[i - 2];
        q[i] = a[i] * q[i - 1] + q[i - 2];
        if(p[i] * p[i] - n * q[i] * q[i] == 1)
            return {p[i], q[i]};
    }
} // $x^2 - n * y^2 = 1$ 最小正整数根，n 为完全平方数时无解
```

Simpson

```
// 三次函数，两倍精度拟合
// $error = \frac{(r-l)^5}{6480}|f^{(4)}|$
// $\int_a^b f(x)\,dx \approx \frac{(b-a)}{8}\left[f(a) + 3f\left(\frac{2a+b}{3}\right) + 3f\left(\frac{a+2b}{3}\right) + f(b)\right]$
// 三次函数拟合 $error = \frac{1}{90}\left(\frac{r-l}{2}\right)^5|f^{(4)}|$
d simpson(d fl,d fr,d fmid,d l,d r) {
    return (fl+fr+4.0*fmid)*(r-l)/6.0; }
d rsimpson(d slr,d fl,d fr,d fmid,d l,d r) {
    d mid = (l+r)/2,fml = f((l+mid)/2),fmr = f((mid+r)/2);
    d slm = simpson(fl,fmid,fml,l,mid);
    d smr = simpson(fmid,fr,fmr,mid,r);
    if(fabs(slr - smr - slm) / slr < eps)return slm + smr;
    return rsimpson(slm,fl,fmid,fml,l,mid)+
        rsimpson(smr,fmid,fr,fmr,mid,r);
}
```

解一元三次方程

听说极端情况精度不够

```
double a(p[3]), b(p[2]), c(p[1]), d(p[0]);
double k(b / a), m(c / a), n(d / a);
double p(-k * k / 3. + m);
```

```
double q(2. * k * k * k / 27 - k * m / 3. + n);
Complex omega[3] = {Complex(1, 0), Complex(-0.5, 0.5 * sqrt(3)), Complex(-0.5, -0.5 *
  ↪ sqrt(3))};
Complex r1, r2;
double delta(q * q / 4 + p * p * p / 27);
if (delta > 0) {
    r1 = cubrt(-q / 2. + sqrt(delta));
    r2 = cubrt(-q / 2. - sqrt(delta));
} else {
    r1 = pow(-q / 2. + pow(Complex(delta), 0.5), 1. / 3);
    r2 = pow(-q / 2. - pow(Complex(delta), 0.5), 1. / 3);
}
for(int _(0); _ < 3; _++) {
    Complex x = -k / 3. + r1 * omega[_ * 1] + r2 * omega[_ * 2 % 3];
}
```

线段下整点

solve for $\sum_{i=0}^{n-1} \lfloor \frac{a+bi}{m} \rfloor$, $n, m, a, b > 0$

```
LL solve(LL n,LL a,LL b,LL m){
    if(b==0) return n*(a/m);
    if(a>=m) return n*(a/m)+solve(n,a%m,b,m);
    if(b>=m) return (n-1)*n/2*(b/m)+solve(n,a,b%m,m);
    return solve((a+b*n)/m,(a+b*n)%m,m,b);
}
```

线性同余不等式

```
// Find the minimal non-negtive solutions for l ≤ d·x mod m ≤ r
// 0 ≤ d,l,r < m; l ≤ r, O(log n)
ll cal(ll m, ll d, ll l, ll r) {
    if (l == 0) return 0;
    if (d == 0) return MXL; // 无解
    if (d * 2 > m) return cal(m, m - d, m - r, m - l);
    if ((l - 1) / d < r / d) return (l - 1) / d + 1;
    ll k = cal(d, (-m % d + d) % d, l % d, r % d);
    return k == MXL ? MXL : (k * m + l - 1) / d + 1; // 无解 2
}
```

EX-BSGS -zzq

```
/*
 * EX_BSGS
```

```
 * a^x = b (mod p)
 * p may not be a prime
 */

ll qpow(ll a, ll x, ll Mod) {
    ll res = 1;
    for (; x; x >>= 1) {
        if (x & 1) res = res * a % Mod;
        a = a * a % Mod;
    }
    return res;
}

std::unordered_map<int, int> mp;

ll exbsgs(ll a, ll b, ll p) {
    if (b == 1) return 0;
    ll t, d = 1, k = 0;
    while ((t = std::__gcd(a, p)) != 1) {
        if (b % t) return -1;
        ++k, b /= t, p /= t, d = d * (a / t) % p;
        if (b == d) return k;
    }
    mp.clear();
    ll m = std::ceil(std::sqrt(p));
    ll a_m = qpow(a, m, p);
    ll mul = b;
    for (ll j = 1; j <= m; ++j) {
        mul = mul * a % p;
        mp[mul] = j;
    }
    for (ll i = 1; i <= m; ++i) {
        d = d * a_m % p;
        if (mp.count(d)) return i * m - mp[d] + k;
    }
    return -1;
}
```

EX-BSGS -zky

```
LL BSGS(LL a,LL b,LL p){
    LL m=sqrt(p)+.5,v=inv(pw(a,m,p),p),e=1;
    map<LL,LL>hash;hash[1]=0;
    for(int i=1;i<m;i++)
        e=e*a%p,hash[e]=i;
```

```
    for(int i=0;i<=m;i++){
        if(hash.count(b))return i*m+hash[b];
        b=b*v%p;
    }return -1;
}

LL solve2(LL a,LL b,LL p){
    //a^x=b (mod p)
    b%=p;
    LL e=1%p;
    for(int i=0;i<100;i++){
        if(e==b)return i;
        e=e*a%p;
    }
    int r=0;
    while(gcd(a,p)!=1){
        LL d=gcd(a,p);
        if(b%d)return -1;
        p/=d;b/=d;b=b*inv(a/d,p);
        r++;
    }LL res=BSGS(a,b,p);
    if(res==-1)return -1;
    return res+r;
}
```

分治乘法

```
(a+b)(c+d) = ac+(bc+ad)+bd = 2ac-(a-b)(c-d)+2bd

x = x^m m=(n+1)/2
(ax+b)(cx+d) = x^2ac + x(bc+ad) + bd = x^2ac + x(ac + bd - (a-b)(c-d)) + bd
```

组合数模 p^k

```
LL prod=1,P;
pair<LL,LL> comput(LL n,LL p,LL k){
    if(n<=1)return make_pair(0,1);
    LL ans=1,cnt=0;
    ans=pow(prod,n/P,P);
    cnt=n/p;
    pair<LL,LL>res=comput(n/p,p,k);
    cnt+=res.first;
    ans=ans*res.second%P;
    for(int i=n-n%P+1;i<=n;i++)if(i%p){
```

```

        ans=ans*i%P;
    }
    return make_pair(cnt,ans);
}
pair<LL,LL> calc(LL n,LL p,LL k){
    prod=1;P=pow(p,k,1e18);
    for(int i=1;i<P;i++)if(i%p)prod=prod*i%P;
    pair<LL,LL> res=comput(n,p,k);
//  res.second=res.second*pow(p,res.first%k,P)%P;
//  res.first-=res.first%k;
    return res;
}
LL calc(LL n,LL m,LL p,LL k){
    pair<LL,LL>A,B,C;
    LL P=pow(p,k,1e18);
    A=calc(n,p,k);
    B=calc(m,p,k);
    C=calc(n-m,p,k);
    LL ans=1;
    ans=pow(p,A.first-B.first-C.first,P);
    ans=ans*A.second%P*inv(B.second,P)%P*inv(C.second,P)%P;
    return ans;
}
```

线性筛

```
void sieve(){
    f[1]=mu[1]=phi[1]=1;
    for(int i=2;i<maxn;i++){
        if(!minp[i]){
            minp[i]=i;
            minpw[i]=i;
            mu[i]=-1;
            phi[i]=i-1;
            f[i]=i-1;
            p[++p[0]]=i;//Case 1 prime
        }
        for(int j=1;j<=p[0]&&(LL)i*p[j]<maxn;j++){
            minp[i*p[j]]=p[j];
            if(i%p[j]==0){
                //Case 2 not coprime
                minpw[i*p[j]]=minpw[i]*p[j];
                phi[i*p[j]]=phi[i]*p[j];
                mu[i*p[j]]=0;
```

```
                if(i==minpw[i]){
                    f[i*p[j]]=i*p[j]-i;//Special Case for f(p^k)
                }else{
                    f[i*p[j]]=f[i/minpw[i]]*f[minpw[i]*p[j]];
                }
                break;
            }else{
                //Case 3 coprime
                minpw[i*p[j]]=p[j];
                f[i*p[j]]=f[i]*f[p[j]];
                phi[i*p[j]]=phi[i]*(p[j]-1);
                mu[i*p[j]]=-mu[i];
            }
        }
    }
}
```

Chapter 3

图论

图论基础

```
struct Graph {  // Remember to call .init()!
    int e, nxt[M], v[M], adj[N], n;
    bool base;
    __inline void init(bool _base, int _n = 0) {
        assert(n < N);
        n = _n; base = _base;
        e = 0; memset(adj + base, -1, sizeof(*adj) * n);
    }
    __inline int new_node() {
        adj[n + base] = -1;
        assert(n + base + 1 < N);
        return n++ + base;
    }
    __inline void ins(int u0, int v0) {  // directional
        assert(u0 < n + base && v0 < n + base);
        v[e] = v0; nxt[e] = adj[u0]; adj[u0] = e++;
        assert(e < M);
    }
    __inline void bi_ins(int u0, int v0) {  // bi-directional
        ins(u0, v0); ins(v0, u0);
    }
};
```

坚固无敌的点双

```
const bool BCC_VERTEX = 0, BCC_EDGE = 1;
struct BCC {  // N = N0 + M0. Remember to call init(&raw_graph).
    Graph *g, forest; // g is raw graph ptr.
    int dfn[N], DFN, low[N];
    int stack[N], top;
```

```
    int expand_to[M];      // Where edge i is expanded to in expaned graph.
    // Vertex i expaned to i.
    int compress_to[N];  // Where vertex i is compressed to.
    bool cut[N], compress_cut[N], branch[M], vis[N], flag;
    //std::vector<int> BCC_component[N];  // Cut vertex belongs to none.
    __inline void init(Graph *raw_graph) {
        g = raw_graph;
    }
    void DFS(int u, int pe) {
        dfn[u] = low[u] = ++DFN; cut[u] = false;
        if (!~g->adj[u]) {
            cut[u] = 1;
            compress_to[u] = forest.new_node();
            compress_cut[compress_to[u]] = 1;
        }
        for (int e = g->adj[u]; ~e; e = g->nxt[e]) {
            int v = g->v[e];
            if ((e ^ pe) > 1 && dfn[v] > 0 && dfn[v] < dfn[u]) {
                stack[top++] = e;
                low[u] = std::min(low[u], dfn[v]);
            }
            else if (!dfn[v]) {
                stack[top++] = e; branch[e] = 1;
                DFS(v, e);
                low[u] = std::min(low[v], low[u]);
                if (low[v] >= dfn[u]) {
                    if ((pe == -1 && flag || pe != -1) && !cut[u]) {
                        cut[u] = 1;
                        compress_to[u] = forest.new_node();
                        compress_cut[compress_to[u]] = 1;
                    }
                    int cc = forest.new_node();
                    if (cut[u]) {
                        forest.bi_ins(compress_to[u], cc);
                    }
                    compress_cut[cc] = 0;
                    //BCC_component[cc].clear();
                    do {
                        int cur_e = stack[--top];
                        compress_to[expand_to[cur_e]] = cc;
                        compress_to[expand_to[cur_e^1]] = cc;
                        if (branch[cur_e]) {
                            int v = g->v[cur_e];
                            if (cut[v]) {
                                forest.bi_ins(cc, compress_to[v]);
                            } else {
```

```
                                //BCC_component[cc].push_back(v);
                                compress_to[v] = cc;
                            }
                        }
                    } while (stack[top] != e);
                    if (pe == -1 && !flag) {
                        compress_to[u] = cc;
                    }
                }
            }
        }
    }
    inline bool dfs(int u, int pe) {
        vis[u] = 1;
        int d = 0;
        for (int e = g->adj[u]; ~e; e = g->nxt[e]) {
            int v = g -> v[e];
            if (!vis[v]) {
                ++d;
                dfs(v, e);
            }
        }
        return pe == -1 ? d > 1 : 0;
    }
    void solve() {
        forest.init(g->base);
        int n = g->n;
        for (int i = 0; i < g->e; i++) {
            expand_to[i] = g->new_node();
        }
        memset(vis + g -> base, 0, sizeof(*vis) * n);
        memset(branch, 0, sizeof(*branch) * g->e);
        memset(dfn + g->base, 0, sizeof(*dfn) * n); DFN = 0;
        for (int i = 0; i < n; i++)
            if (!dfn[i + g->base]) {
                top = 0;
                flag = dfs(i + g -> base, -1);
                DFS(i + g->base, -1);
            }
    }
} bcc;
```

坚固无敌的边双

```
struct BCC {
    Graph *g, forest;
    int dfn[N], low[N], stack[N], tot[N], belong[N], vis[N], top, dfs_clock;
    // tot[] is the size of each BCC, belong[] is the BCC that each node belongs to
    pair<int, int > ori[M]; // bridge in raw_graph(raw node)
    bool is_bridge[M];
    __inline void init(Graph *raw_graph) {
        g = raw_graph;
        memset(is_bridge, false, sizeof(*is_bridge) * g -> e);
        memset(vis + g -> base, 0, sizeof(*vis) * g -> n);
    }
    void tarjan(int u, int from) {
        dfn[u] = low[u] = ++dfs_clock; vis[u] = 1; stack[++top] = u;
        for (int p = g -> adj[u]; ~p; p = g -> nxt[p]) {
            if ((p ^ 1) == from) continue;
            int v = g -> v[p];
            if (vis[v]) {
                if (vis[v] == 1) low[u] = min(low[u], dfn[v]);
            } else {
                tarjan(v, p);
                low[u] = min(low[u], low[v]);
                if (low[v] > dfn[u]) is_bridge[p / 2] = true;
            }
        }
        if (dfn[u] != low[u]) return;
        tot[forest.new_node()] = 0;
        do {
            belong[stack[top]] = forest.n;
            vis[stack[top]] = 2;
            tot[forest.n]++;
            --top;
        } while (stack[top + 1] != u);
    }
    void solve() {
        forest.init(g -> base);
        int n = g -> n;
        for (int i = 0; i < n; ++i)
            if (!vis[i + g -> base]) {
                top = dfs_clock = 0;
                tarjan(i + g -> base, -1);
            }
        for (int i = 0; i < g -> e / 2; ++i)
            if (is_bridge[i]) {
                int e = forest.e;
```

```
                forest.bi_ins(belong[g -> v[i * 2]], belong[g -> v[i * 2 + 1]], g -> w[i *
  ↪ 2]);
                ori[e] = make_pair(g -> v[i * 2 + 1], g -> v[i * 2]);
                ori[e + 1] = make_pair(g -> v[i * 2], g -> v[i * 2 + 1]);
            }
        }
} bcc;
```

闪电二分图匹配

```
int matchx[N], matchy[N], level[N];
vector<int> edge[N];
bool dfs(int x) {
    for (int i = 0; i < (int)edge[x].size(); ++i) {
        int y = edge[x][i];
        int w = matchy[y];
        if (w == -1 || level[x] + 1 == level[w] && dfs(w)) {
            matchx[x] = y;
            matchy[y] = x;
            return true;
        }
    }
    level[x] = -1;
    return false;
}
int solve() {
    memset(matchx, -1, sizeof(*matchx) * n);
    memset(matchy, -1, sizeof(*matchy) * m);
    for (int ans = 0; ; ) {
        std::vector<int> q;
        for (int i = 0; i < n; ++i) {
            if (matchx[i] == -1) {
                level[i] = 0;
                q.push_back(i);
            } else {
                level[i] = -1;
            }
        }
        for (int head = 0; head < (int)q.size(); ++head) {
            int x = q[head];
            for (int i = 0; i < (int)edge[x].size(); ++i) {
                int y = edge[x][i];
                int w = matchy[y];
                if (w != -1 && level[w] < 0) {
                    level[w] = level[x] + 1;
```

```
                    q.push_back(w);
                }
            }
        }
        int delta = 0;
        for (int i = 0; i < n; ++i) {
            if (matchx[i] == -1 && dfs(i)) {
                delta++;
            }
        }
        if (delta == 0) {
            return ans;
        } else {
            ans += delta;
        }
    }
}
```

一般图匹配

```
// 0-base, match[u] is linked to u
vector<int> lnk[MAXN];
int match[MAXN], Queue[MAXN], pred[MAXN], base[MAXN], head, tail, sta, fin, nbase;
bool inQ[MAXN], inB[MAXN];
inline void push(int u) {
    Queue[tail++] = u; inQ[u] = 1;
}
inline int pop() {
    return Queue[head++];
}
inline int FindCA(int u, int v) {
    static bool inP[MAXN];
    fill(inP, inP + n, false);
    while (1) {
        u = base[u]; inP[u] = 1;
        if(u == sta) break;
        u = pred[match[u]];
    }
    while (1) {
        v = base[v];
        if (inP[v]) break;
        v = pred[match[v]];
    }
    return v;
}
```

```
inline void RT(int u) {
    int v;
    while (base[u] != nbase) {
        v = match[u];
        inB[base[u]] = inB[base[v]] = 1;
        u = pred[v];
        if (base[u] != nbase) pred[u] = v;
    }
}
inline void BC(int u, int v) {
    nbase = FindCA(u, v);
    fill(inB, inB + n, 0);
    RT(u); RT(v);
    if (base[u] != nbase) pred[u] = v;
    if (base[v] != nbase) pred[v] = u;
    for (int i = 0; i < n; ++i)
        if (inB[base[i]]) {
            base[i] = nbase;
            if (!inQ[i]) push(i);
        }
}
bool FindAP(int u) {
    bool found = false;
    for (int i = 0; i < n; ++i) {
        pred[i] = -1; base[i] = i; inQ[i] = 0;
    }
    sta = u; fin = -1; head = tail = 0; push(sta);
    while (head < tail) {
        int u = pop();
        for (int i = (int)lnk[u].size() - 1; i >= 0; --i) {
            int v = lnk[u][i];
            if (base[u] != base[v] && match[u] != v) {
                if (v == sta || match[v] >= 0 && pred[match[v]] >= 0) BC(u, v);
                else if (pred[v] == -1) {
                    pred[v] = u;
                    if (match[v] >= 0) push(match[v]);
                    else {
                        fin = v;
                        return true;
                    }
                }
            }
        }
    }
    return found;
}
```

```
inline void AP() {
    int u = fin, v, w;
    while (u >= 0) {
        v = pred[u]; w = match[v];
        match[v] = u; match[u] = v;
        u = w;
    }
}
inline int FindMax() {
    for (int i = 0; i < n; ++i) match[i] = -1;
    for (int i = 0; i < n; ++i)
        if (match[i] == -1 && FindAP(i)) AP();
    int ans = 0;
    for (int i = 0; i < n; ++i) {
        ans += (match[i] != -1);
    }
    return ans;
}
```

一般最大权匹配

```
//maximum weight blossom,  change g[u][v].w to INF - g[u][v].w when minimum weight blossom
  ↪ is needed
//type of ans is long long
//replace all int to long long if weight of edge is long long

struct WeightGraph {
    static const int INF = INT_MAX;
    static const int MAXN = 400;
    struct edge{
        int u, v, w;
        edge() {}
        edge(int u, int v, int w): u(u), v(v), w(w) {}
    };
    int n, n_x;
    edge g[MAXN * 2 + 1][MAXN * 2 + 1];
    int lab[MAXN * 2 + 1];
    int match[MAXN * 2 + 1], slack[MAXN * 2 + 1], st[MAXN * 2 + 1], pa[MAXN * 2 + 1];
    int flower_from[MAXN * 2 + 1][MAXN+1], S[MAXN * 2 + 1], vis[MAXN * 2 + 1];
    vector<int> flower[MAXN * 2 + 1];
    queue<int> q;
    inline int e_delta(const edge &e){ // does not work inside blossoms
        return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
    }
    inline void update_slack(int u, int x){
```

```
        if(!slack[x] || e_delta(g[u][x]) < e_delta(g[slack[x]][x]))
            slack[x] = u;
    }
    inline void set_slack(int x){
        slack[x] = 0;
        for(int u = 1;u <= n; ++u)
            if(g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
                update_slack(u, x);
    }
    void q_push(int x){
        if(x <= n)q.push(x);
        else for(size_t i = 0;i < flower[x].size(); i++)
            q_push(flower[x][i]);
    }
    inline void set_st(int x, int b){
        st[x]=b;
        if(x > n) for(size_t i = 0;i < flower[x].size(); ++i)
                set_st(flower[x][i], b);
    }
    inline int get_pr(int b, int xr){
        int pr = find(flower[b].begin(), flower[b].end(), xr) - flower[b].begin();
        if(pr % 2 == 1){
            reverse(flower[b].begin() + 1, flower[b].end());
            return (int)flower[b].size() - pr;
        } else return pr;
    }
    inline void set_match(int u, int v){
        match[u]=g[u][v].v;
        if(u > n){
            edge e=g[u][v];
            int xr = flower_from[u][e.u], pr=get_pr(u, xr);
            for(int i = 0;i < pr; ++i)
                set_match(flower[u][i], flower[u][i ^ 1]);
            set_match(xr, v);
            rotate(flower[u].begin(), flower[u].begin()+pr, flower[u].end());
        }
    }
    inline void augment(int u, int v){
        for(; ; ){
            int xnv=st[match[u]];
            set_match(u, v);
            if(!xnv)return;
            set_match(xnv, st[pa[xnv]]);
            u=st[pa[xnv]], v=xnv;
        }
    }
```

```
    inline int get_lca(int u, int v){
        static int t=0;
        for(++t; u || v; swap(u, v)){
            if(u == 0)continue;
            if(vis[u] == t)return u;
            vis[u] = t;
            u = st[match[u]];
            if(u) u = st[pa[u]];
        }
        return 0;
    }
    inline void add_blossom(int u, int lca, int v){
        int b = n + 1;
        while(b <= n_x && st[b]) ++b;
        if(b > n_x) ++n_x;
        lab[b] = 0, S[b] = 0;
        match[b] = match[lca];
        flower[b].clear();
        flower[b].push_back(lca);
        for(int x = u, y; x != lca; x = st[pa[y]]) {
            flower[b].push_back(x),
            flower[b].push_back(y = st[match[x]]),
            q_push(y);
        }
        reverse(flower[b].begin() + 1, flower[b].end());
        for(int x = v, y; x != lca; x = st[pa[y]]) {
            flower[b].push_back(x),
            flower[b].push_back(y = st[match[x]]),
            q_push(y);
        }
        set_st(b, b);
        for(int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w = 0;
        for(int x = 1; x <= n; ++x) flower_from[b][x] = 0;
        for(size_t i = 0 ; i < flower[b].size(); ++i){
            int xs = flower[b][i];
            for(int x = 1; x <= n_x; ++x)
                if(g[b][x].w == 0 || e_delta(g[xs][x]) < e_delta(g[b][x]))
                    g[b][x] = g[xs][x], g[x][b] = g[x][xs];
            for(int x = 1;x <= n; ++x)
                if(flower_from[xs][x]) flower_from[b][x] = xs;
        }
        set_slack(b);
    }
    inline void expand_blossom(int b){ // S[b]  ==  1
        for(size_t i = 0; i < flower[b].size(); ++i)
            set_st(flower[b][i], flower[b][i]);
```

```
        int xr = flower_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
        for(int i = 0; i < pr; i += 2){
            int xs = flower[b][i], xns = flower[b][i + 1];
            pa[xs] = g[xns][xs].u;
            S[xs] = 1, S[xns] = 0;
            slack[xs] = 0, set_slack(xns);
            q_push(xns);
        }
        S[xr] = 1, pa[xr] = pa[b];
        for(size_t i = pr + 1;i < flower[b].size(); ++i){
            int xs = flower[b][i];
            S[xs] = -1, set_slack(xs);
        }
        st[b] = 0;
    }
    inline bool on_found_edge(const edge &e){
        int u = st[e.u], v = st[e.v];
        if(S[v] == -1){
            pa[v] = e.u, S[v] = 1;
            int nu = st[match[v]];
            slack[v] = slack[nu] = 0;
            S[nu] = 0, q_push(nu);
        }else if(S[v] == 0){
            int lca = get_lca(u, v);
            if(!lca) return augment(u, v), augment(v, u), true;
            else add_blossom(u, lca, v);
        }
        return false;
    }
    inline bool matching(){
        memset(S + 1, -1, sizeof(int) * n_x);
        memset(slack + 1, 0, sizeof(int) * n_x);
        q = queue<int>();
        for(int x = 1;x <= n_x; ++x)
            if(st[x] == x && !match[x]) pa[x]=0, S[x]=0, q_push(x);
        if(q.empty())return false;
        for(;;){
            while(q.size()){
                int u = q.front();q.pop();
                if(S[st[u]] == 1)continue;
                for(int v = 1;v <= n; ++v)
                    if(g[u][v].w > 0 && st[u] != st[v]){
                        if(e_delta(g[u][v]) == 0){
                            if(on_found_edge(g[u][v]))return true;
                        }else update_slack(u, st[v]);
                    }
```

```
            }
            int d = INF;
            for(int b = n + 1; b <= n_x;++b)
                if(st[b] == b && S[b] == 1)d = min(d, lab[b]/2);
            for(int x = 1; x <= n_x; ++x)
                if(st[x] == x && slack[x]){
                    if(S[x] == -1)d = min(d, e_delta(g[slack[x]][x]));
                    else if(S[x] == 0)d = min(d, e_delta(g[slack[x]][x])/2);
                }
            for(int u = 1; u <= n; ++u){
                if(S[st[u]] == 0){
                    if(lab[u] <= d)return 0;
                    lab[u] -= d;
                }else if(S[st[u]] == 1)lab[u] += d;
            }
            for(int b = n+1; b <= n_x; ++b)
                if(st[b] == b){
                    if(S[st[b]] == 0) lab[b] += d * 2;
                    else if(S[st[b]] == 1) lab[b] -= d * 2;
                }
            q=queue<int>();
            for(int x = 1; x <= n_x; ++x)
                if(st[x] == x && slack[x] && st[slack[x]] != x && e_delta(g[slack[x]][x]) ==
    ↪ 0)
                    if(on_found_edge(g[slack[x]][x]))return true;
            for(int b = n + 1; b <= n_x; ++b)
                if(st[b] == b && S[b] == 1 && lab[b] == 0)expand_blossom(b);
        }
        return false;
    }
    inline pair<long long, int> solve(){
        memset(match + 1, 0, sizeof(int) * n);
        n_x = n;
        int n_matches = 0;
        long long tot_weight = 0;
        for(int u = 0; u <= n; ++u) st[u] = u, flower[u].clear();
        int w_max = 0;
        for(int u = 1; u <= n; ++u)
            for(int v = 1; v <= n; ++v){
                flower_from[u][v] = (u == v ? u : 0);
                w_max = max(w_max, g[u][v].w);
            }
        for(int u = 1; u <= n; ++u) lab[u] = w_max;
        while(matching()) ++n_matches;
        for(int u = 1; u <= n; ++u)
            if(match[u] && match[u] < u)
```

```
                tot_weight += g[u][match[u]].w;
        return make_pair(tot_weight, n_matches);
    }
    inline void init(){
        for(int u = 1; u <= n; ++u)
            for(int v = 1; v <= n; ++v)
                g[u][v]=edge(u, v, 0);
    }
};
```

无向图最小割

```
/*
 * Stoer Wagner 全局最小割 O(V ^ 3)
 * 1base, 点数 n, 邻接矩阵 edge[MAXN][MAXN]
 * 返回值为全局最小割
 */

int StoerWagner() {
    static int v[MAXN], wage[MAXN];
    static bool vis[MAXN];

    for (int i = 1; i <= n; ++i) v[i] = i;

    int res = INF;

    for (int nn = n; nn > 1; --nn) {
        memset(vis, 0, sizeof(bool) * (nn + 1));
        memset(wage, 0, sizeof(int) * (nn + 1));

        int pre, last = 1; // vis[1] = 1;

        for (int i = 1; i < nn; ++i) {
            pre = last; last = 0;
            for (int j = 2; j <= nn; ++j) if (!vis[j]) {
                wage[j] += edge[v[pre]][v[j]];
                if (!last || wage[j] > wage[last]) last = j;
            }
            vis[last] = 1;
        }

        res = std::min(res, wage[last]);

        for (int i = 1; i <= nn; ++i) {
            edge[v[i]][v[pre]] += edge[v[last]][v[i]];
```

```
            edge[v[pre]][v[i]] += edge[v[last]][v[i]];
        }
        v[last] = v[nn];
    }
    return res;
}
```

最大带权带花树

```
//maximum weight blossom,  change g[u][v].w to INF - g[u][v].w when minimum weight blossom
  ↪ is needed
//type of ans is long long
//replace all int to long long if weight of edge is long long

struct WeightGraph {
    static const int INF = INT_MAX;
    static const int MAXN = 400;
    struct edge{
        int u, v, w;
        edge() {}
        edge(int u, int v, int w): u(u), v(v), w(w) {}
    };
    int n, n_x;
    edge g[MAXN * 2 + 1][MAXN * 2 + 1];
    int lab[MAXN * 2 + 1];
    int match[MAXN * 2 + 1], slack[MAXN * 2 + 1], st[MAXN * 2 + 1], pa[MAXN * 2 + 1];
    int flower_from[MAXN * 2 + 1][MAXN+1], S[MAXN * 2 + 1], vis[MAXN * 2 + 1];
    vector<int> flower[MAXN * 2 + 1];
    queue<int> q;
    inline int e_delta(const edge &e){ // does not work inside blossoms
        return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
    }
    inline void update_slack(int u, int x){
        if(!slack[x] || e_delta(g[u][x]) < e_delta(g[slack[x]][x]))
            slack[x] = u;
    }
    inline void set_slack(int x){
        slack[x] = 0;
        for(int u = 1;u <= n; ++u)
            if(g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
                update_slack(u, x);
    }
    void q_push(int x){
        if(x <= n)q.push(x);
        else for(size_t i = 0;i < flower[x].size(); i++)
```

```
            q_push(flower[x][i]);
    }
    inline void set_st(int x, int b){
        st[x]=b;
        if(x > n) for(size_t i = 0;i < flower[x].size(); ++i)
                    set_st(flower[x][i], b);
    }
    inline int get_pr(int b, int xr){
        int pr = find(flower[b].begin(), flower[b].end(), xr) - flower[b].begin();
        if(pr % 2 == 1){
            reverse(flower[b].begin() + 1, flower[b].end());
            return (int)flower[b].size() - pr;
        } else return pr;
    }
    inline void set_match(int u, int v){
        match[u]=g[u][v].v;
        if(u > n){
            edge e=g[u][v];
            int xr = flower_from[u][e.u], pr=get_pr(u, xr);
            for(int i = 0;i < pr; ++i)
                set_match(flower[u][i], flower[u][i ^ 1]);
            set_match(xr, v);
            rotate(flower[u].begin(), flower[u].begin()+pr, flower[u].end());
        }
    }
    inline void augment(int u, int v){
        for(; ; ){
            int xnv=st[match[u]];
            set_match(u, v);
            if(!xnv)return;
            set_match(xnv, st[pa[xnv]]);
            u=st[pa[xnv]], v=xnv;
        }
    }
    inline int get_lca(int u, int v){
        static int t=0;
        for(++t; u || v; swap(u, v)){
            if(u == 0)continue;
            if(vis[u] == t)return u;
            vis[u] = t;
            u = st[match[u]];
            if(u) u = st[pa[u]];
        }
        return 0;
    }
    inline void add_blossom(int u, int lca, int v){
```

```
        int b = n + 1;
        while(b <= n_x && st[b]) ++b;
        if(b > n_x) ++n_x;
        lab[b] = 0, S[b] = 0;
        match[b] = match[lca];
        flower[b].clear();
        flower[b].push_back(lca);
        for(int x = u, y; x != lca; x = st[pa[y]]) {
            flower[b].push_back(x),
            flower[b].push_back(y = st[match[x]]),
            q_push(y);
        }
        reverse(flower[b].begin() + 1, flower[b].end());
        for(int x = v, y; x != lca; x = st[pa[y]]) {
            flower[b].push_back(x),
            flower[b].push_back(y = st[match[x]]),
            q_push(y);
        }
        set_st(b, b);
        for(int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w = 0;
        for(int x = 1; x <= n; ++x) flower_from[b][x] = 0;
        for(size_t i = 0 ; i < flower[b].size(); ++i){
            int xs = flower[b][i];
            for(int x = 1; x <= n_x; ++x)
                if(g[b][x].w == 0 || e_delta(g[xs][x]) < e_delta(g[b][x]))
                    g[b][x] = g[xs][x], g[x][b] = g[x][xs];
            for(int x = 1;x <= n; ++x)
                if(flower_from[xs][x]) flower_from[b][x] = xs;
        }
        set_slack(b);
    }
    inline void expand_blossom(int b){ // S[b]  ==  1
        for(size_t i = 0; i < flower[b].size(); ++i)
            set_st(flower[b][i], flower[b][i]);
        int xr = flower_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
        for(int i = 0; i < pr; i += 2){
            int xs = flower[b][i], xns = flower[b][i + 1];
            pa[xs] = g[xns][xs].u;
            S[xs] = 1, S[xns] = 0;
            slack[xs] = 0, set_slack(xns);
            q_push(xns);
        }
        S[xr] = 1, pa[xr] = pa[b];
        for(size_t i = pr + 1;i < flower[b].size(); ++i){
            int xs = flower[b][i];
            S[xs] = -1, set_slack(xs);
```

```
        }
        st[b] = 0;
    }
    inline bool on_found_edge(const edge &e){
        int u = st[e.u], v = st[e.v];
        if(S[v] == -1){
            pa[v] = e.u, S[v] = 1;
            int nu = st[match[v]];
            slack[v] = slack[nu] = 0;
            S[nu] = 0, q_push(nu);
        }else if(S[v] == 0){
            int lca = get_lca(u, v);
            if(!lca) return augment(u, v), augment(v, u), true;
            else add_blossom(u, lca, v);
        }
        return false;
    }
    inline bool matching(){
        memset(S + 1, -1, sizeof(int) * n_x);
        memset(slack + 1, 0, sizeof(int) * n_x);
        q = queue<int>();
        for(int x = 1;x <= n_x; ++x)
            if(st[x] == x && !match[x]) pa[x]=0, S[x]=0, q_push(x);
        if(q.empty())return false;
        for(;;){
            while(q.size()){
                int u = q.front();q.pop();
                if(S[st[u]] == 1)continue;
                for(int v = 1;v <= n; ++v)
                    if(g[u][v].w > 0 && st[u] != st[v]){
                        if(e_delta(g[u][v]) == 0){
                            if(on_found_edge(g[u][v]))return true;
                        }else update_slack(u, st[v]);
                    }
            }
            int d = INF;
            for(int b = n + 1; b <= n_x;++b)
                if(st[b] == b && S[b] == 1)d = min(d, lab[b]/2);
            for(int x = 1; x <= n_x; ++x)
                if(st[x] == x && slack[x]){
                    if(S[x] == -1)d = min(d, e_delta(g[slack[x]][x]));
                    else if(S[x] == 0)d = min(d, e_delta(g[slack[x]][x])/2);
                }
            for(int u = 1; u <= n; ++u){
                if(S[st[u]] == 0){
                    if(lab[u] <= d)return 0;
```

```
                    lab[u] -= d;
                }else if(S[st[u]] == 1)lab[u] += d;
            }
            for(int b = n+1; b <= n_x; ++b)
                if(st[b] == b){
                    if(S[st[b]] == 0) lab[b] += d * 2;
                    else if(S[st[b]] == 1) lab[b] -= d * 2;
                }
            q=queue<int>();
            for(int x = 1; x <= n_x; ++x)
                if(st[x] == x && slack[x] && st[slack[x]] != x && e_delta(g[slack[x]][x]) ==
 ↪ 0)
                    if(on_found_edge(g[slack[x]][x]))return true;
            for(int b = n + 1; b <= n_x; ++b)
                if(st[b] == b && S[b] == 1 && lab[b] == 0)expand_blossom(b);
        }
        return false;
    }
    inline pair<long long, int> solve(){
        memset(match + 1, 0, sizeof(int) * n);
        n_x = n;
        int n_matches = 0;
        long long tot_weight = 0;
        for(int u = 0; u <= n; ++u) st[u] = u, flower[u].clear();
        int w_max = 0;
        for(int u = 1; u <= n; ++u)
            for(int v = 1; v <= n; ++v){
                flower_from[u][v] = (u == v ? u : 0);
                w_max = max(w_max, g[u][v].w);
            }
        for(int u = 1; u <= n; ++u) lab[u] = w_max;
        while(matching()) ++n_matches;
        for(int u = 1; u <= n; ++u)
            if(match[u] && match[u] < u)
                tot_weight += g[u][match[u]].w;
        return make_pair(tot_weight, n_matches);
    }
    inline void init(){
        for(int u = 1; u <= n; ++u)
            for(int v = 1; v <= n; ++v)
                g[u][v]=edge(u, v, 0);
    }
};
```

必经点 Dominator-tree

```
//solve(s, n, raw_g): s is the root and base accords to base of raw_g
//idom[x] will be x if x does not have a dominator,and will be -1 if x is not reachable from
  ↪ s.

struct dominator_tree {
    int base, dfn[N], sdom[N], idom[N], id[N], f[N], fa[N], smin[N], stamp;
    Graph *g;
    void predfs(int u) {
        id[dfn[u] = stamp++] = u;
        for (int i = g -> adj[u]; ~i; i = g -> nxt[i]) {
            int v = g -> v[i];
            if (dfn[v] < 0) {
                f[v] = u;
                predfs(v);
            }
        }
    }
    int getfa(int u) {
        if (fa[u] == u) return u;
        int ret = getfa(fa[u]);
        if (dfn[sdom[smin[fa[u]]]] < dfn[sdom[smin[u]]])
            smin[u] = smin[fa[u]];
        return fa[u] = ret;
    }
    void solve (int s, int n, Graph *raw_graph) {
        g = raw_graph;
        base = g -> base;
        memset(dfn + base, -1, sizeof(*dfn) * n);
        memset(idom + base, -1, sizeof(*idom) * n);
        static Graph pred, tmp;
        pred.init(base, n);
        for (int i = 0; i < n; ++i) {
            for (int p = g -> adj[i + base]; ~p; p = g -> nxt[p])
                pred.ins(g -> v[p], i + base);
        }
        stamp = 0; tmp.init(base, n); predfs(s);
        for (int i = 0; i < stamp; ++i) {
            fa[id[i]] = smin[id[i]] = id[i];
        }
        for (int o = stamp - 1; o >= 0; --o) {
            int x = id[o];
            if (o) {
                sdom[x] = f[x];
                for (int i = pred.adj[x]; ~i; i = pred.nxt[i]) {
```

```
                    int p = pred.v[i];
                    if (dfn[p] < 0) continue;
                    if (dfn[p] > dfn[x]) {
                        getfa(p);
                        p = sdom[smin[p]];
                    }
                    if (dfn[sdom[x]] > dfn[p]) sdom[x] = p;
                }
                tmp.ins(sdom[x], x);
            }
            while (~tmp.adj[x]) {
                int y = tmp.v[tmp.adj[x]];
                tmp.adj[x] = tmp.nxt[tmp.adj[x]];
                getfa(y);
                if (x != sdom[smin[y]]) idom[y] = smin[y];
                else idom[y] = x;
            }
            for (int i = g -> adj[x]; ~i; i = g -> nxt[i])
                if (f[g -> v[i]] == x) fa[g -> v[i]] = x;
        }
        idom[s] = s;
        for (int i = 1; i < stamp; ++i) {
            int x = id[i];
            if (idom[x] != sdom[x]) idom[x] = idom[idom[x]];
        }
    }
};
```

欧拉回路

```
//从一个奇度点 dfs，sqn 即为回路/路径
//first 存点，second 存边的编号，正反边编号一致
//清空 cur、used 数组
void getCycle(int u)
{
    for(int &i=cur[u]; i < (int)adj[u].size(); ++ i) {
        int id = adj[u][i].second;
        if (used[id]) continue;
        used[id] = true;
        getCycle(adj[u][i].first);
    }
    sqn.push_back(u);
}
```

朱刘最小树形图

```cpp
struct D_MT {
    struct Edge {
        int u, v, w;
        inline Edge() {}
        inline Edge(int _u, int _v, int _w):u(_u), v(_v), w(_w) {
        }
    };
    int nn, mm, n, m, vis[maxn], pre[maxn], id[maxn], in[maxn];
    Edge edges[maxn], bac[maxn];
    void init(int _n) {
        n = _n;
        m = 0;
    }
    void AddEdge(int u, int v, int w) {
        edges[m++] = Edge(u, v, w);
    }
    int work(int root) {
        int ret = 0;
        while(true) {
            for (int i = 0; i < n; i++) in[i]=inf + 1;
            for (int i = 0; i < m; i++) {
                int u = edges[i].u, v = edges[i].v;
                if(edges[i].w < in[v] && u != v){
                    in[v] = edges[i].w;
                    pre[v] = u;
                }
            }
            for (int i = 0; i < n; i++) {
                if(i == root) continue;
                if(in[i] == inf + 1) return inf;
            }
            int cnt = 0;
            for (int i = 0; i < n; i++) {
                id[i] = -1;
                vis[i] = -1;
            }
            in[root] = 0;
            for (int i = 0; i < n; i++) {
                ret += in[i];
                int v = i;
                while (vis[v] != i&& id[v] == -1 && v != root ){
                    vis[v] = i;
                    v = pre[v];
                }
```

```
                if (v != root && id[v] == -1) {
                    for (int u = pre[v]; u != v; u = pre[u]) id[u] = cnt;
                    id[v] = cnt++;
                }
            }
            if (!cnt) break;
            for (int i=0; i<n; i++)
                if (id[i] == -1) id[i] = cnt++;
            for (int i = 0; i < m; i++){
                int u = edges[i].u, v = edges[i].v;
                edges[i].v = id[v];
                edges[i].u = id[u];
                if(id[u] != id[v]) edges[i].w -= in[v];
            }
            n = cnt;
            root = id[root];
        }
        return ret;
    }
} MT;
```

Chapter 4

数据结构

Kd-tree

```
int n;
LL norm(const LL &x) {
    //    For manhattan distance
       //return std::abs(x);
    //    For euclid distance
    return x * x;
}

struct P{
    int a[2],val;
    int id;
    int& operator[](int s){return a[s];}
    const int& operator[](int s)const{return a[s];}

    LL dis(const P &b)const{
        LL ans=0;
        for (int i = 0; i < 2; ++i) {
            ans += norm(a[i] - b[i]);
        }
        return ans;
    }
}p[maxn];

bool operator==(const P &a,const P &b){
    for(int i=0;i<DIM;i++)
        if(a[i]!=b[i])
            return false;
    return true;
}
bool byVal(P a,P b){
    return a.val!=b.val ? a.val<b.val : a.id<b.id;
```

```
}

struct Rec{
    int mn[DIM],mx[DIM];
    Rec(){}
    Rec(const P &p){
        for(int i=0;i<DIM;i++){
            mn[i]=mx[i]=p[i];
        }
    }
    void add(const P &p){
        for(int i=0;i<DIM;i++){
            mn[i]=min(p[i],mn[i]);
            mx[i]=max(p[i],mx[i]);
        }
    }

    LL dis(const P &p) {
        LL ans = 0;
        for (int i = 0; i < 2; ++i) {
            //    For minimum distance
            ans += norm(min(max(p[i], mn[i]), mx[i]) - p[i]);
            //    For maximum distance
            //ans += std::max(norm(max[i] - p[i]), norm(min[i] - p[i]));
        }
        return ans;
    }
};
inline Rec operator+(const Rec &ls,const Rec &rs){
    static Rec rec;
    for(int i=0;i<DIM;i++){
        rec.mn[i]=min(ls.mn[i],rs.mn[i]);
        rec.mx[i]=max(ls.mx[i],rs.mx[i]);
    }
    return rec;
}
struct node{
    Rec rec;
    P sep;
    int sum,siz;
    node *c[2];
    node *rz(){
        sum=sep.val;
        rec=Rec(sep);
        siz=1;
        if(c[0]){
```

```
            sum+=c[0]->sum;
            rec=rec+c[0]->rec;
            siz+=c[0]->siz;
        }
        if(c[1]){
            sum+=c[1]->sum;
            rec=rec+c[1]->rec;
            siz+=c[1]->siz;
        }
        return this;
    }
    node(){sum=0;siz=1;c[0]=c[1]=0;}
}*root,*re,pool[maxn],*cur=pool;
node *sta[maxn];
P tmp[maxn];
int D,si;
void init(){
    si=0;
    cur=pool;
    root=0;
}
bool cmp(const P &A,const P &B){

    if(!(A[D]==B[D]))
        return A[D]<B[D];
    return A.id<B.id;
}
int top;
node *newnode(){
    if(si)return sta[si--];
    return cur++;
}
node* build(P *p,int l,int r,int d){
    int mid=(l+r)>>1;D=d;
    nth_element(p+l,p+mid,p+r+1,cmp);
    node *t=newnode();
    t->sep=p[mid];
    if(l<=mid-1)
        t->c[0]=build(p,l,mid-1,d^1);
    if(mid+1<=r)
        t->c[1]=build(p,mid+1,r,d^1);
    return t->rz();
}
void dfs(node *&t){
    if(t->c[0])dfs(t->c[0]);
    tmp[++top]=t->sep;
```

```
    if(t->c[1])dfs(t->c[1]);
    sta[++si]=t;*t=node();
    //delete t;
}
node* rebuild(node *&t){
    if(!t)return 0;
    top=0;dfs(t);
    return build(tmp,1,top,0);
}
#define siz(x) (x?x->siz:0)
void Add(node *&t,const P &p,int d=0){//调用前 re=0; 调用后 rebuild(re);
    D=d;
    if(!t){
        t=newnode();
        t->sep=p;t->rz();
        return;
    }
    if(t->sep==p){
        t->sep.val+=p.val;
        t->rz();
        return;
    }
    if(p[D]<t->sep[D])
        Add(t->c[0],p,d^1);
    else
        Add(t->c[1],p,d^1);

    t->rz();

    if(max(siz(t->c[0]),siz(t->c[1]))>0.7*t->siz)
        re=t;
}
int ans;

bool Out(const Rec &a,const Rec &b){
    for(int i=0;i<DIM;i++){
        int l=max(a.mn[i],b.mn[i]);
        int r=min(a.mx[i],b.mx[i]);
        if(l>r)
            return true;
    }
    return false;
}
bool In(const Rec &a,const Rec &b){
    for(int i=0;i<DIM;i++){
        if(a.mn[i]<b.mn[i])
```

```
            return false;
        if(a.mx[i]>b.mx[i])
            return false;
    }
    return true;
}

bool In(const P &a,const Rec &b){
    for(int i=0;i<DIM;i++){
        if(!(b.mn[i]<=a[i]&&a[i]<=b.mx[i]))
            return false;
    }
    return true;
}

void Q(node *t,const Rec &R){
    if(Out(t->rec,R))return ;
    if(In(t->rec,R)){
        ans+=t->sum;
        return;
    }
    if(In(t->sep,R))
        ans+=t->sep.val;
    if(t->c[0])
        Q(t->c[0],R);
    if(t->c[1])
        Q(t->c[1],R);
}

priority_queue<pair<long long, int> > kNN;
void query(node *t, const P &p, int k, int d = 0) {//用钱清空 kNN
    D=d;
    if (!t || ((int)kNN.size() == k && t->rec.dis(p) > kNN.top().first)) {
        return;
    }
    kNN.push(make_pair(t->sep.dis(p), t->sep.id));
    if ((int)kNN.size() > k) {
        kNN.pop();
    }
    if (cmp(p, t->sep)) {
        query(t->c[0], p, k, d ^ 1);
        query(t->c[1], p, k, d ^ 1);
    } else {
        query(t->c[1], p, k, d ^ 1);
        query(t->c[0], p, k, d ^ 1);
    }
```

```
}
```

LCT

```
struct LCT{
    struct node{
        bool rev;
        int mx,val;
        node *f,*c[2];
        bool d(){return this==f->c[1];}
        bool rt(){return !f||(f->c[0]!=this&&f->c[1]!=this);}
        void sets(node *x,int d){pd();if(x)x->f=this;c[d]=x;rz();}
        void makerv(){rev^=1;swap(c[0],c[1]);}
        void pd(){
            if(rev){
                if(c[0])c[0]->makerv();
                if(c[1])c[1]->makerv();
                rev=0;
            }
        }
        void rz(){
            mx=val;
            if(c[0])mx=max(mx,c[0]->mx);
            if(c[1])mx=max(mx,c[1]->mx);
        }
    }nd[int(1e4)+1];
    void rot(node *x){
        node *y=x->f;if(!y->rt())y->f->pd();
        y->pd();x->pd();bool d=x->d();
        y->sets(x->c[!d],d);
        if(y->rt())x->f=y->f;
        else y->f->sets(x,y->d());
        x->sets(y,!d);
    }
    void splay(node *x){
        while(!x->rt())
            if(x->f->rt())rot(x);
            else if(x->d()==x->f->d())rot(x->f),rot(x);
            else rot(x),rot(x);
    }
    node* access(node *x){
        node *y=0;
        for(;x;x=x->f){
            splay(x);
            x->sets(y,1);y=x;
```

```
        }return y;
    }
    void makert(node *x){
        access(x)->makerv();
        splay(x);
    }
    void link(node *x,node *y){
        makert(x);
        x->f=y;
        access(x);
    }
    void cut(node *x,node *y){
        makert(x);access(y);splay(y);
        y->c[0]=x->f=0;
        y->rz();
    }
    void link(int x,int y){link(nd+x,nd+y);}
    void cut(int x,int y){cut(nd+x,nd+y);}
}T;
```

树状数组上二分第 k 大

```
int find(int k){
    int cnt=0,ans=0;
    for(int i=22;i>=0;i--){
        ans+=(1<<i);
        if(ans>n || cnt+d[ans]>=k)ans-=(1<<i);
        else cnt+=d[ans];
    }
    return ans+1;
}
```

Treap

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=1e5+5;
#define sz(x) (x?x->siz:0)
struct Treap{
    struct node{
        int key,val;
        int siz,s;
        node *c[2];
        node(int v=0){
```

```
            val=v;
            key=rand();
            siz=1,s=1;
            c[0]=c[1]=0;
        }
        void rz(){siz=s;if(c[0])siz+=c[0]->siz;if(c[1])siz+=c[1]->siz;}
    }pool[maxn],*cur,*root;
    Treap(){cur=pool;}
    node* newnode(int val){return *cur=node(val),cur++;}
    void rot(node *&t,int d){
        if(!t->c[d])t=t->c[!d];
        else{
            node *p=t->c[d];t->c[d]=p->c[!d];
            p->c[!d]=t;t->rz();p->rz();t=p;
        }
    }
    void insert(node *&t,int x){
        if(!t){t=newnode(x);return;}
        if(t->val==x){t->s++;t->siz++;return;}
        insert(t->c[x>t->val],x);
        if(t->key<t->c[x>t->val]->key)
            rot(t,x>t->val);
        else t->rz();
    }
    void del(node *&t,int x){
        if(!t)return;
        if(t->val==x){
            if(t->s>1){t->s--;t->siz--;return;}
            if(!t->c[0]||!t->c[1]){
                if(!t->c[0])t=t->c[1];
                else t=t->c[0];
                return;
            }
            int d=t->c[0]->key<t->c[1]->key;
            rot(t,d);
            del(t,x);
            return;
        }
        del(t->c[x>t->val],x);
        t->rz();
    }
    int pre(node *t,int x){
        if(!t)return INT_MIN;
        int ans=pre(t->c[x>t->val],x);
        if(t->val<x)ans=max(ans,t->val);
        return ans;
```

```
    }
    int nxt(node *t,int x){
        if(!t)return INT_MAX;
        int ans=nxt(t->c[x>=t->val],x);
        if(t->val>x)ans=min(ans,t->val);
        return ans;
    }
    int rank(node *t,int x){
        if(!t)return 0;
        if(t->val==x)return sz(t->c[0]);
        if(t->val<x)return sz(t->c[0])+t->s+rank(t->c[1],x);
        if(t->val>x)return rank(t->c[0],x);
    }
    int kth(node *t,int x){
        if(sz(t->c[0])>=x)return kth(t->c[0],x);
        if(sz(t->c[0])+t->s>=x)return t->val;
        return kth(t->c[1],x-t->s-sz(t->c[0]));
    }
    void deb(node *t){
        if(!t)return;
        deb(t->c[0]);
        printf("%d ",t->val);
        deb(t->c[1]);
    }
    void insert(int x){insert(root,x);}
    void del(int x){del(root,x);}
    int pre(int x){return pre(root,x);}
    int nxt(int x){return nxt(root,x);}
    int rank(int x){return rank(root,x);}
    int kth(int x){return kth(root,x);}
    void deb(){deb(root);puts("");}
}T;
```

FHQ-Treap

```
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
const int maxn=1e5+5;
int in(){
    int r=0,f=1;char c=getchar();
    while(!isdigit(c))f=c=='-'?-1:f,c=getchar();
    while(isdigit(c))r=r*10+c-'0',c=getchar();
    return r*f;
}
```

```
int n,m;
#define sz(x) (x?x->siz:0)
struct node{
    int siz,key;
    LL val,sum;
    LL mu,a,d;
    node *c[2],*f;
    void split(int ned,node *&p,node *&q);
    node* rz(){
        sum=val;siz=1;
        if(c[0])sum+=c[0]->sum,siz+=c[0]->siz;
        if(c[1])sum+=c[1]->sum,siz+=c[1]->siz;
        return this;
    }
    void make(LL _mu,LL _a,LL _d){
        sum=sum*_mu+_a*siz+_d*siz*(siz-1)/2;
        val=val*_mu+_a+_d*sz(c[0]);
        mu*=_mu;a=a*_mu+_a;d=d*_mu+_d;
    }
    void pd(){
        if(mu==1&&a==0&&d==0)return;
        if(c[0])c[0]->make(mu,a,d);
        if(c[1])c[1]->make(mu,a+d+d*sz(c[0]),d);
        mu=1;a=d=0;
    }
    node(){mu=1;}
}nd[maxn*2],*root;
node *merge(node *p,node *q){
    if(!p||!q)return p?p->rz():(q?q->rz():0);
    p->pd();q->pd();
    if(p->key<q->key){
        p->c[1]=merge(p->c[1],q);
        return p->rz();
    }else{
        q->c[0]=merge(p,q->c[0]);
        return q->rz();
    }
}
void node::split(int ned,node *&p,node *&q){
    if(!ned){p=0;q=this;return;}
    if(ned==siz){p=this;q=0;return;}
    pd();
    if(sz(c[0])>=ned){
        c[0]->split(ned,p,q);c[0]=0;rz();
        q=merge(q,this);
    }else{
```

```
        c[1]->split(ned-sz(c[0])-1,p,q);c[1]=0;rz();
        p=merge(this,p);
    }
}
int tot;
void C(int l,int r,int v){
    node *p,*q,*x,*y;
    root->split(l-1,p,q);
    q->split(r-l+1,x,y);
    x->make(0,v,0);x->pd();
    root=merge(p,merge(x,y));
}
void A(int l,int r,int d){
    node *p,*q,*x,*y;
    root->split(l-1,p,q);
    q->split(r-l+1,x,y);
    x->make(1,d,d);x->pd();
    root=merge(p,merge(x,y));
}
void I(int ps,int v){
    node *p,*q;
    root->split(ps-1,p,q);
    node *x=nd+(++tot);
    x->key=rand();x->val=v;x->rz();
    root=merge(merge(p,x),q);
}
LL Q(int l,int r){
    node *p,*q,*x,*y;
    root->split(l-1,p,q);
    q->split(r-l+1,x,y);
    LL ans=x->sum;
    root=merge(p,merge(x,y));
    return ans;
}
int main(){
//  freopen("bzoj3188.in","r",stdin);
    n=in();m=in();
    for(int i=1;i<=n;i++){
        nd[i].val=in();
        nd[i].key=rand();
        nd[i].rz();
        root=merge(root,nd+i);
    }tot=n;
    while(m--){
        int ty=in();
        int l,r;
```

```
        if(ty==1){
            l=in();r=in();
            C(l,r,in());
        }else if(ty==2){
            l=in();r=in();
            A(l,r,in());
        }else if(ty==3){
            int ps=in();
            I(ps,in());
        }else if(ty==4){
            l=in();r=in();
            printf("%lld\n",Q(l,r));
        }
    }
    return 0;
}
```

真-FHQTreap

```
const int mo=1e9+7;
int rnd(){
    static int x=1;
    return x=(x*23333+233);
}
int rnd(int n){
    int x=rnd();
    if(x<0)x=-x;
    return x%n+1;
}
struct node{
    int siz,key;
    int val;
    LL sum;
    node *c[2];
    node* rz(){
        sum=val;siz=1;
        if(c[0])sum+=c[0]->sum,siz+=c[0]->siz;
        if(c[1])sum+=c[1]->sum,siz+=c[1]->siz;
        return this;
    }
    node(){}
    node(int v){
        siz=1;key=rnd();
        val=v;sum=v;
        c[0]=c[1]=0;
```

```
    }

}pool[maxn*8],*root,*cur=pool,*old_root,*stop;
node *newnode(int v=0){
    *cur=node(v);
    return cur++;
}
node *old_merge(node *p,node *q){
    if(!p&&!q)return 0;
    node *u=0;
    if(!p||!q)return u=p?p->rz():(q?q->rz():0);
    if(rnd(sz(p)+sz(q))<sz(p)){
        u=p;
        u->c[1]=old_merge(u->c[1],q);
    }else{
        u=q;
        u->c[0]=old_merge(p,u->c[0]);
    }
    return u->rz();
}
node *merge(node *p,node *q){
    if(!p&&!q)return 0;
    node *u=newnode();
    if(!p||!q)return u=p?p->rz():(q?q->rz():0);
    if(rnd(sz(p)+sz(q))<sz(p)){
        *u=*p;
        u->c[1]=merge(u->c[1],q);
    }else{
        *u=*q;
        u->c[0]=merge(p,u->c[0]);
    }
    return u->rz();
}
node *split(node *u,int l,int r){
    if(l>r||!u)return 0;
    node *x=0;
    if(l==1&&r==sz(u)){
        x=newnode();
        *x=*u;
        return x->rz();
    }
    int lsz=sz(u->c[0]);
    if(r<=lsz)
        return split(u->c[0],l,r);
    if(l>lsz+1)
        return split(u->c[1],l-lsz-1,r-lsz-1);
```

```
    x=newnode();
    *x=*u;
    x->c[0]=split(u->c[0],l,lsz);
    x->c[1]=split(u->c[1],1,r-lsz-1);
    return x->rz();
}
```

莫队上树

```
bool operator<(qes a,qes b){
    if(dfn[a.x]/B!=dfn[b.x]/B)return dfn[a.x]/B<dfn[b.x]/B;
    if(dfn[a.y]/B!=dfn[b.y]/B)return dfn[a.y]/B<dfn[b.y]/B;
    if(a.tm/B!=b.tm/B)return a.tm/B<b.tm/B;
    return a.tm<b.tm;
}
void vxor(int x){
    if(vis[x])ans-=(LL)W[cnt[col[x]]]*V[col[x]],cnt[col[x]]--;
    else cnt[col[x]]++,ans+=(LL)W[cnt[col[x]]]*V[col[x]];
    vis[x]^=1;
}
void change(int x,int y){
    if(vis[x]){
        vxor(x);col[x]=y;vxor(x);
    }else col[x]=y;
}
void TimeMachine(int tar){//XD
    for(int i=now+1;i<=tar;i++)change(C[i].x,C[i].y);
    for(int i=now;i>tar;i--)change(C[i].x,C[i].pre);
    now=tar;
}
void vxor(int x,int y){
    while(x!=y)if(dep[x]>dep[y])vxor(x),x=fa[x];
    else vxor(y),y=fa[y];
}
    for(int i=1;i<=q;i++){
        int ty=getint(),x=getint(),y=getint();
        if(ty&&dfn[x]>dfn[y])swap(x,y);
        if(ty==0) C[++Csize]=(oper){x,y,pre[x],i},pre[x]=y;
        else Q[Qsize+1]=(qes){x,y,Qsize+1,Csize},Qsize++;
    }sort(Q+1,Q+1+Qsize);
    int u=Q[1].x,v=Q[1].y;
    TimeMachine(Q[1].tm);
    vxor(Q[1].x,Q[1].y);
    int LCA=lca(Q[1].x,Q[1].y);
    vxor(LCA);anss[Q[1].id]=ans;vxor(LCA);
```

```
    for(int i=2;i<=Qsize;i++){
        TimeMachine(Q[i].tm);
        vxor(Q[i-1].x,Q[i].x);
        vxor(Q[i-1].y,Q[i].y);
        int LCA=lca(Q[i].x,Q[i].y);
        vxor(LCA);
        anss[Q[i].id]=ans;
        vxor(LCA);
    }
```

虚树

```
int a[maxn*2],sta[maxn*2];
int top=0,k;
void build(){
    top=0;
    sort(a,a+k,bydfn);
    k=unique(a,a+k)-a;
    sta[top++]=1;_n=k;
    for(int i=0;i<k;i++){
        int LCA=lca(a[i],sta[top-1]);
        while(dep[LCA]<dep[sta[top-1]]){
            if(dep[LCA]>=dep[sta[top-2]]){
                add_edge(LCA,sta[--top]);
                if(sta[top-1]!=LCA)sta[top++]=LCA;
                break;
            }add_edge(sta[top-2],sta[top-1]);top--;
        }if(sta[top-1]!=a[i])sta[top++]=a[i];
    }
    while(top>1)
        add_edge(sta[top-2],sta[top-1]),top--;
    for(int i=0;i<k;i++)inr[a[i]]=1;
}
```

Chapter 5

字符串

Manacher

```
//prime is the origin string(0-base)
//-10,-1,-20 are added to s
//length of s is exactly 2 * l + 3
inline void manacher(char prime[]) {
    int l = strlen(prime), n = 0;
    s[n++] = -10;
    s[n++] = -1;
    for (int i = 0; i < l; ++i) {
        s[n++] = prime[i];
        s[n++] = -1;
    }
    s[n++] = -20; f[0] = 1;
    int mx = 0, id = 0;
    for (int i = 1; i + 1 < n; ++i) {
        f[i] = i > mx ? 1 : min(f[id * 2 - i], mx - i + 1);
        while (s[i + f[i]] == s[i - f[i]]) ++f[i];
        if (i + f[i] - 1 > mx) {
            mx = i + f[i] - 1;
            id = i;
        }
    }
}
```

指针版回文自动机

```
/*
 * Palindrome Automaton - pointer version
 * PAMPAMPAM? PAMPAMPAM!
 */

```

```
namespace PAM {
    struct Node *pool_pointer;
    struct Node {
        Node *fail, *to[26];
        int cnt, len;

        Node() {}
        Node(int len): len(len) {
            memset(to, 0, sizeof(to));
            fail = 0;
            cnt = 0;
        }

        void *operator new (size_t) {
            return pool_pointer++;
        }
    } pool[100005], *root[2], *last;
    int pam_len, str[100005];

    void init() {
        pool_pointer = pool;
        root[0] = new Node(0);
        root[1] = new Node(-1);
        root[0]->fail = root[1]->fail = root[1];
        str[pam_len = 0] = -1; // different from all characters
        last = root[0];
    }

    void extend(char ch) {
        static Node *p, *np, *q;

        int x = str[++pam_len] = ch - 'a';

        p = last;
        while (str[pam_len - p->len - 1] != x)
            p = p->fail;
        if (!p->to[x]) {
            np = new Node(p->len + 2), q = p->fail;
            while (str[pam_len - q->len - 1] != x) q = q->fail;
            np->fail = q->to[x] ? q->to[x] : root[0];
            p->to[x] = np;
        }
        last = p->to[x];
        ++last->cnt;
    }
}
```

数组版后缀自动机

```cpp
/*
 * Suffix Automaton - array version
 * SAMSAMSAM? SAMSAMSAM!
 */

namespace SAM {
    int to[100005 << 1][26], parent[100005 << 1], step[100005 << 1], tot;
    int root, np;
    int sam_len;

    int newnode(int STEP = 0) {
        ++tot;
        memset(to[tot], 0, sizeof to[tot]);
        parent[tot] = 0;
        step[tot] = STEP;
        return tot;
    }

    void init() {
        tot = 0;
        root = np = newnode(sam_len = 0);
    }

    void extend(char ch) {
        int x = ch - 'a';
        int last = np; np = newnode(++sam_len);
        for (; last && !to[last][x]; last = parent[last])
            to[last][x] = np;
        if (!last) parent[np] = root;
        else {
            int q = to[last][x];
            if (step[q] == step[last] + 1) parent[np] = q;
            else {
                nq = newnode(step[last] + 1);
                memcpy(to[nq], to[q], sizeof to[q]);
                parent[nq] = parent[q];
                parent[q] = parent[np] = nq;
                for (; last && to[last][x] == q; last = parent[last])
                    to[last][x] = nq;
            }
        }
    }
}
```

指针版后缀自动机

```
/*
 * Suffix Automaton - pointer version
 * SAMSAMSAM? SAMSAMSAM!
 */

namespace SAM {
    struct Node *pool_pointer;
    struct Node {
        Node *to[26], *parent;
        int step;

        Node(int STEP = 0): step(STEP) {
            memset(to, 0, sizeof to);
            parent = 0;
            step = 0;
        }

        void *operator new (size_t) {
            return pool_pointer++;
        }
    } pool[100005 << 1], *root, *np;
    int sam_len;

    void init() {
        pool_pointer = pool;
        root = np = new Node(sam_len = 0);
    }

    void extend(char ch) {
        static Node *last, *q, *nq;

        int x = ch - 'a';
        last = np; np = new Node(++sam_len);
        for (; last && !last->to[x]; last = last->parent)
            last->to[x] = np;
        if (!last) np->parent = root;
        else {
            q = last->to[x];
            if (q->step == last->step + 1) np->parent = q;
            else {
                nq = new Node(*q);
                nq->step = last->step + 1;
                q->parent = np->parent = nq;
                for (; last && last->to[x] == q; last = last->parent)
```

```
                    last->to[x] = nq;
                }
            }
        }
}
```

广义后缀自动机

```
/*
 * EX Suffix Automaton - pointer version
 * SAMSAMSAM? SAMSAMSAM!
 */

namespace SAM {
    struct Node *pool_pointer;
    struct Node {
        Node *parent, *to[26];
        int step;

        Node(int step = 0): step(step) {
            memset(to, 0, sizeof to);
            parent = 0;
        }

        void *operator new (size_t) {
            return pool_pointer++;
        }
    } pool[100005 * 10 << 1], *root, *np;
    int sam_len, now_len;

    void init() {
        sam_len = now_len = 0;
        pool_pointer = pool;
        root = new Node();
    }

    void new_str() { // a new string start
        now_len = 0;
        np = root;
    }

    void extend(char ch) {
        static Node *last, *q, *nq;

        int x = ch - 'a';
```

```
        if (np->to[x]) {
            np = np->to[x];
            ++now_len;
        }
        else {
            last = np; np = new Node(++now_len);
            for (; last && !last->to[x]; last = last->parent)
                last->to[x] = np;
            if (!last) np->parent = root;
            else {
                q = last->to[x];
                if (q->step == last->step + 1) np->parent = q;
                else {
                    nq = new Node(*q);
                    nq->step = last->step + 1;
                    q->parent = np->parent = nq;
                    for (; last && last->to[x] == q; last = last->parent)
                        last->to[x] = nq;
                }
            }
        }

        sam_len = std::max(sam_len, now_len);
    }
}
```

后缀数组

```
const int maxl=1e5+1e4+5;
const int maxn=maxl*2;
int a[maxn],x[maxn],y[maxn],c[maxn],sa[maxn],rank[maxn],height[maxn];
void calc_sa(int n){
    int m=alphabet,k=1;
    memset(c,0,sizeof(*c)*(m+1));
    for(int i=1;i<=n;i++)c[x[i]=a[i]]++;
    for(int i=1;i<=m;i++)c[i]+=c[i-1];
    for(int i=1;i<=n;i++)sa[c[x[i]]--]=i;
    for(;k<=n;k<<=1){
        int tot=k;
        for(int i=n-k+1;i<=n;i++)y[i-n+k]=i;
        for(int i=1;i<=n;i++)
            if(sa[i]>k)y[++tot]=sa[i]-k;
        memset(c,0,sizeof(*c)*(m+1));
        for(int i=1;i<=n;i++)c[x[i]]++;
        for(int i=1;i<=m;i++)c[i]+=c[i-1];
```

```
        for(int i=n;i>=1;i--)sa[c[x[y[i]]]--]=y[i];
        for(int i=1;i<=n;i++)y[i]=x[i];
        tot=1;x[sa[1]]=1;
        for(int i=2;i<=n;i++){
            if(max(sa[i],sa[i-1])+k>n||y[sa[i]]!=y[sa[i-1]]||y[sa[i]+k]!=y[sa[i-1]+k])
                ++tot;
            x[sa[i]]=tot;
        }
        if(tot==n)break;else m=tot;
    }
}
void calc_height(int n){
    for(int i=1;i<=n;i++)rank[sa[i]]=i;
    for(int i=1;i<=n;i++){
        height[rank[i]]=max(0,height[rank[i-1]]-1);
        if(rank[i]==1)continue;
        int j=sa[rank[i]-1];
        while(max(i,j)+height[rank[i]]<=n&&a[i+height[rank[i]]]==a[j+height[rank[i]]])
            ++height[rank[i]];
    }
}
```

最小表示法

```
int solve(char *text, int length) {//0-base , 多解答案为起点最小
    int i = 0, j = 1, delta = 0;
    while (i < length && j < length && delta < length) {
        char tokeni = text[(i + delta) % length];
        char tokenj = text[(j + delta) % length];
        if (tokeni == tokenj) {
            delta++;
        } else {
            if (tokeni > tokenj) {
                i += delta + 1;
            } else {
                j += delta + 1;
            }
            if (i == j) {
                j++;
            }
            delta = 0;
        }
    }
    return std::min(i, j);
}
```

Chapter 6

计算几何

点类

```
int sgn(double x){return (x>eps)-(x<-eps);}
int sgn(double a,double b){return sgn(a-b);}
double sqr(double x){return x*x;}
struct P{
    double x,y;
    P(){}
    P(double x,double y):x(x),y(y){}
    double len2(){
        return sqr(x)+sqr(y);
    }
    double len(){
        return sqrt(len2());
    }
    void print(){
        printf("(%.3f,%.3f)\n",x,y);
    }
    P turn90(){return P(-y,x);}
    P norm(){return P(x/len(),y/len());}
};
bool operator==(P a,P b){
    return !sgn(a.x-b.x) and !sgn(a.y-b.y);
}
P operator+(P a,P b){
    return P(a.x+b.x,a.y+b.y);
}
P operator-(P a,P b){
    return P(a.x-b.x,a.y-b.y);
}
P operator*(P a,double b){
    return P(a.x*b,a.y*b);
}
```

```
P operator/(P a,double b){
    return P(a.x/b,a.y/b);
}
double operator^(P a,P b){
    return a.x*b.x + a.y*b.y;
}
double operator*(P a,P b){
    return a.x*b.y - a.y*b.x;
}
double det(P a,P b,P c){
    return (b-a)*(c-a);
}
double dis(P a,P b){
    return (b-a).len();
}
double Area(vector<P>poly){
    double ans=0;
    for(int i=1;i<poly.size();i++)
        ans+=(poly[i]-poly[0])*(poly[(i+1)%poly.size()]-poly[0]);
    return fabs(ans)/2;
}
struct L{
    P a,b;
    L(){}
    L(P a,P b):a(a),b(b){}
    P v(){return b-a;}
};
bool onLine(P p,L l){
    return sgn((l.a-p)*(l.b-p))==0;
}
bool onSeg(P p,L s){
    return onLine(p,s) and sgn((s.b-s.a)^(p-s.a))>=0 and sgn((s.a-s.b)^(p-s.b))>=0;
}
bool parallel(L l1,L l2){
    return sgn(l1.v()*l2.v())==0;
}
P intersect(L l1,L l2){
    double s1=det(l1.a,l1.b,l2.a);
    double s2=det(l1.a,l1.b,l2.b);
    return (l2.a*s2-l2.b*s1)/(s2-s1);
}
P project(P p,L l){
    return l.a+l.v()*((p-l.a)^l.v())/l.v().len2();
}
double dis(P p,L l){
    return fabs((p-l.a)*l.v())/l.v().len();
```

```
}
int dir(P p,L l){
    int t=sgn((p-l.b)*(l.b-l.a));
    if(t<0)return -1;
    if(t>0)return 1;
    return 0;
}
bool segIntersect(L l1,L l2){//strictly
    if(dir(l2.a,l1)*dir(l2.b,l1)<0&&dir(l1.a,l2)*dir(l1.b,l2)<0)
        return true;
    return false;
}
bool in_tri(P pt,P *p){
    if((p[1]-p[0])*(p[2]-p[0])<0)
        reverse(p,p+3);
    for(int i=0;i<3;i++){
        if(!onLeft(pt,L(p[i],p[(i+1)%3])))
            return false;
    }
    return true;
}
```

圆基础

```
struct C{
    P o;
    double r;
    C(){}
    C(P _o,double _r):o(_o),r(_r){}
};
// 求圆与直线的交点
//turn90() P(-y,x)
double fix(double x){return x>=0?x:0;}
bool intersect(C a, L l, P &p1, P &p2) {
    double x = ((l.a - a.o)^ (l.b - l.a)),
        y = (l.b - l.a).len2(),
        d = x * x - y * ((l.a - a.o).len2() - a.r * a.r);
    if (sgn(d) < 0) return false;
    d = max(d, 0.0);
    P p = l.a - ((l.b - l.a) * (x / y)), delta = (l.b - l.a) * (sqrt(d) / y);
    p1 = p + delta, p2 = p - delta;
    return true;
}
// 求圆与圆的交点，注意调用前要先判定重圆
bool intersect(C a, C b, P &p1, P &p2) {
```

```
    double s1 = (a.o - b.o).len();
    if (sgn(s1 - a.r - b.r) > 0 || sgn(s1 - fabs(a.r - b.r)) < 0) return false;
    double s2 = (a.r * a.r - b.r * b.r) / s1;
    double aa = (s1 + s2) * 0.5, bb = (s1 - s2) * 0.5;
    P o = (b.o - a.o) * (aa / (aa + bb)) + a.o;
    P delta = (b.o - a.o).norm().turn90() * sqrt(fix(a.r * a.r - aa * aa));
    p1 = o + delta, p2 = o - delta;
    return true;
}
// 求点到圆的切点，按关于点的顺时针方向返回两个点
bool tang(const C &c, const P &p0, P &p1, P &p2) {
    double x = (p0 - c.o).len2(), d = x - c.r * c.r;
    if (d < eps) return false; // 点在圆上认为没有切点
    P p = (p0 - c.o) * (c.r * c.r / x);
    P delta = ((p0 - c.o) * (-c.r * sqrt(d) / x)).turn90();
    p1 = c.o + p + delta;
    p2 = c.o + p - delta;
    return true;
}
// 求圆到圆的外共切线，按关于 c1.o 的顺时针方向返回两条线
vector<L> extan(const C &c1, const C &c2) {
    vector<L> ret;
    if (sgn(c1.r - c2.r) == 0) {
        P dir = c2.o - c1.o;
        dir = (dir * (c1.r / dir.len())).turn90();
        ret.push_back(L(c1.o + dir, c2.o + dir));
        ret.push_back(L(c1.o - dir, c2.o - dir));
    } else {
        P p = (c1.o * -c2.r + c2.o * c1.r) / (c1.r - c2.r);
        P p1, p2, q1, q2;
        if (tang(c1, p, p1, p2) && tang(c2, p, q1, q2)) {
            if (c1.r < c2.r) swap(p1, p2), swap(q1, q2);
            ret.push_back(L(p1, q1));
            ret.push_back(L(p2, q2));
        }
    }
    return ret;
}
// 求圆到圆的内共切线，按关于 c1.o 的顺时针方向返回两条线
vector<L> intan(const C &c1, const C &c2) {
    vector<L> ret;
    P p = (c1.o * c2.r + c2.o * c1.r) / (c1.r + c2.r);
    P p1, p2, q1, q2;
    if (tang(c1, p, p1, p2) && tang(c2, p, q1, q2)) { // 两圆相切认为没有切线
        ret.push_back(L(p1, q1));
        ret.push_back(L(p2, q2));
```

```
    }
    return ret;
}
```

点在多边形内

```
bool InPoly(P p,vector<P>poly){
    int cnt=0;
    for(int i=0;i<poly.size();i++){
        P a=poly[i],b=poly[(i+1)%poly.size()];
        if(OnLine(p,L(a,b)))
            return false;
        int x=sgn(det(a,p,b));
        int y=sgn(a.y-p.y);
        int z=sgn(b.y-p.y);
        cnt+=(x>0&&y<=0&&z>0);
        cnt-=(x<0&&z<=0&&y>0);
    }
    return cnt;
}
```

二维最小覆盖圆

```
struct line{
    point p,v;
};
point Rev(point v){return point(-v.y,v.x);}
point operator*(line A,line B){
    point u=B.p-A.p;
    double t=(B.v*u)/(B.v*A.v);
    return A.p+A.v*t;
}
point get(point a,point b){
    return (a+b)/2;
}
point get(point a,point b,point c){
    if(a==b)return get(a,c);
    if(a==c)return get(a,b);
    if(b==c)return get(a,b);
    line ABO=(line){(a+b)/2,Rev(a-b)};
    line BCO=(line){(c+b)/2,Rev(b-c)};
    return ABO*BCO;
}
int main(){
```

```
    scanf("%d",&n);
    for(int i=1;i<=n;i++)scanf("%lf%lf",&p[i].x,&p[i].y);
    random_shuffle(p+1,p+1+n);
    O=p[1];r=0;
    for(int i=2;i<=n;i++){
        if(dis(p[i],O)<r+1e-6)continue;
        O=get(p[1],p[i]);r=dis(O,p[i]);
        for(int j=1;j<i;j++){
            if(dis(p[j],O)<r+1e-6)continue;
            O=get(p[i],p[j]);r=dis(O,p[i]);
            for(int k=1;k<j;k++){
                if(dis(p[k],O)<r+1e-6)continue;
                O=get(p[i],p[j],p[k]);r=dis(O,p[i]);
            }
        }
    }printf("%.2lf %.2lf %.2lf\n",O.x,O.y,r);
    return 0;
}s
```

半平面交

```
struct P{
    int quad() const { return sgn(y) == 1 || (sgn(y) == 0 && sgn(x) >= 0);}
};
struct L{
    bool onLeft(const P &p) const { return sgn((b - a)*( p - a)) > 0; }
    L push() const{ // push out eps
        const double eps = 1e-10;
        P delta = (b - a).turn90().norm() * eps;
        return L(a - delta, b - delta);
    }
};
bool sameDir(const L &l0, const L &l1) {
    return parallel(l0, l1) && sgn((l0.b - l0.a)^(l1.b - l1.a)) == 1;
}
bool operator < (const P &a, const P &b) {
    if (a.quad() != b.quad())
        return a.quad() < b.quad();
    else
        return sgn((a*b)) > 0;
}
bool operator < (const L &l0, const L &l1) {
    if (sameDir(l0, l1))
        return l1.onLeft(l0.a);
    else
```

```
        return (l0.b - l0.a) < (l1.b - l1.a);
}
bool check(const L &u, const L &v, const L &w) {
    return w.onLeft(intersect(u, v));
}
vector<P> intersection(vector<L> &l) {
    sort(l.begin(), l.end());
    deque<L> q;
    for (int i = 0; i < (int)l.size(); ++i) {
        if (i && sameDir(l[i], l[i - 1])) {
            continue;
        }
        while (q.size() > 1
            && !check(q[q.size() - 2], q[q.size() - 1], l[i]))
                q.pop_back();
        while (q.size() > 1
            && !check(q[1], q[0], l[i]))
                q.pop_front();
        q.push_back(l[i]);
    }
    while (q.size() > 2
        && !check(q[q.size() - 2], q[q.size() - 1], q[0]))
            q.pop_back();
    while (q.size() > 2
        && !check(q[1], q[0], q[q.size() - 1]))
            q.pop_front();
    vector<P> ret;
    for (int i = 0; i < (int)q.size(); ++i)
    ret.push_back(intersect(q[i], q[(i + 1) % q.size()]));
    return ret;
}
```

求凸包

```
vector<P> convex(vector<P>p){
    sort(p.begin(),p.end());
    vector<P>ans,S;
    for(int i=0;i<p.size();i++){
        while(S.size()>=2
                && sgn(det(S[S.size()-2],S.back(),p[i]))<=0)
                    S.pop_back();
        S.push_back(p[i]);
    }//dw
    ans=S;
    S.clear();
```

```
    for(int i=(int)p.size()-1;i>=0;i--){
        while(S.size()>=2
                && sgn(det(S[S.size()-2],S.back(),p[i]))<=0)
                    S.pop_back();
        S.push_back(p[i]);
    }//up
    for(int i=1;i+1<S.size();i++)
        ans.push_back(S[i]);
    return ans;
}
```

凸包游戏

```
/*
   给定凸包, log n 内完成各种询问, 具体操作有 :
   1. 判定一个点是否在凸包内
   2. 询问凸包外的点到凸包的两个切点
   3. 询问一个向量关于凸包的切点
   4. 询问一条直线和凸包的交点
   INF 为坐标范围, 需要定义点类大于号
   改成实数只需修改 sign 函数, 以及把 long long 改为 double 即可
   构造函数时传入凸包要求无重点, 面积非空, 以及 pair(x,y) 的最小点放在第一个
*/
const int INF = 1000000000;
struct Convex
{
    int n;
    vector<Point> a, upper, lower;
    Convex(vector<Point> _a) : a(_a) {
        n = a.size();
        int ptr = 0;
        for(int i = 1; i < n; ++ i) if (a[ptr] < a[i]) ptr = i;
        for(int i = 0; i <= ptr; ++ i) lower.push_back(a[i]);
        for(int i = ptr; i < n; ++ i) upper.push_back(a[i]);
        upper.push_back(a[0]);
    }
    int sign(long long x) { return x < 0 ? -1 : x > 0; }
    pair<long long, int> get_tangent(vector<Point> &convex, Point vec) {
        int l = 0, r = (int)convex.size() - 2;
        for( ; l + 1 < r; ) {
            int mid = (l + r) / 2;
            if (sign((convex[mid + 1] - convex[mid]).det(vec)) > 0) r = mid;
            else l = mid;
        }
        return max(make_pair(vec.det(convex[r]), r)
```

```
            , make_pair(vec.det(convex[0]), 0));
    }
    void update_tangent(const Point &p, int id, int &i0, int &i1) {
        if ((a[i0] - p).det(a[id] - p) > 0) i0 = id;
        if ((a[i1] - p).det(a[id] - p) < 0) i1 = id;
    }
    void binary_search(int l, int r, Point p, int &i0, int &i1) {
        if (l == r) return;
        update_tangent(p, l % n, i0, i1);
        int sl = sign((a[l % n] - p).det(a[(l + 1) % n] - p));
        for( ; l + 1 < r; ) {
            int mid = (l + r) / 2;
            int smid = sign((a[mid % n] - p).det(a[(mid + 1) % n] - p));
            if (smid == sl) l = mid;
            else r = mid;
        }
        update_tangent(p, r % n, i0, i1);
    }
    int binary_search(Point u, Point v, int l, int r) {
        int sl = sign((v - u).det(a[l % n] - u));
        for( ; l + 1 < r; ) {
            int mid = (l + r) / 2;
            int smid = sign((v - u).det(a[mid % n] - u));
            if (smid == sl) l = mid;
            else r = mid;
        }
        return l % n;
    }
    // 判定点是否在凸包内，在边界返回 true
    bool contain(Point p) {
        if (p.x < lower[0].x || p.x > lower.back().x) return false;
        int id = lower_bound(lower.begin(), lower.end()
            , Point(p.x, -INF)) - lower.begin();
        if (lower[id].x == p.x) {
            if (lower[id].y > p.y) return false;
        } else if ((lower[id - 1] - p).det(lower[id] - p) < 0) return false;
        id = lower_bound(upper.begin(), upper.end(), Point(p.x, INF)
            , greater<Point>()) - upper.begin();
        if (upper[id].x == p.x) {
            if (upper[id].y < p.y) return false;
        } else if ((upper[id - 1] - p).det(upper[id] - p) < 0) return false;
        return true;
    }
    // 求点 p 关于凸包的两个切点，如果在凸包外则有序返回编号
    // 共线的多个切点返回任意一个，否则返回 false
    bool get_tangent(Point p, int &i0, int &i1) {
```

```
        if (contain(p)) return false;
        i0 = i1 = 0;
        int id = lower_bound(lower.begin(), lower.end(), p) - lower.begin();
        binary_search(0, id, p, i0, i1);
        binary_search(id, (int)lower.size(), p, i0, i1);
        id = lower_bound(upper.begin(), upper.end(), p
            , greater<Point>()) - upper.begin();
        binary_search((int)lower.size() - 1, (int)lower.size() - 1 + id, p, i0, i1);
        binary_search((int)lower.size() - 1 + id
            , (int)lower.size() - 1 + (int)upper.size(), p, i0, i1);
        return true;
    }
    // 求凸包上和向量 vec 叉积最大的点，返回编号，共线的多个切点返回任意一个
    int get_tangent(Point vec) {
        pair<long long, int> ret = get_tangent(upper, vec);
        ret.second = (ret.second + (int)lower.size() - 1) % n;
        ret = max(ret, get_tangent(lower, vec));
        return ret.second;
    }
    // 求凸包和直线 u,v 的交点，如果无严格相交返回 false.
    //如果有则是和 (i,next(i)) 的交点，两个点无序，交在点上不确定返回前后两条线段其中之一
    bool get_intersection(Point u, Point v, int &i0, int &i1) {
        int p0 = get_tangent(u - v), p1 = get_tangent(v - u);
        if (sign((v - u).det(a[p0] - u)) * sign((v - u).det(a[p1] - u)) < 0) {
            if (p0 > p1) swap(p0, p1);
            i0 = binary_search(u, v, p0, p1);
            i1 = binary_search(u, v, p1, p0 + n);
            return true;
        } else {
            return false;
        }
    }
};
```

平面最近点

```
bool byY(P a,P b){return a.y<b.y;}
LL solve(P *p,int l,int r){
    LL d=1LL<<62;
    if(l==r)
        return d;
    if(l+1==r)
        return dis2(p[l],p[r]);
    int mid=(l+r)>>1;
    d=min(solve(l,mid),d);
```

```
    d=min(solve(mid+1,r),d);
    vector<P>tmp;
    for(int i=l;i<=r;i++)
        if(sqr(p[mid].x-p[i].x)<=d)
            tmp.push_back(p[i]);
    sort(tmp.begin(),tmp.end(),byY);
    for(int i=0;i<tmp.size();i++)
        for(int j=i+1;j<tmp.size()&&j-i<10;j++)
            d=min(d,dis2(tmp[i],tmp[j]));
    return d;
}
```

Chapter 7

技巧

无敌的读入优化

```
// getchar() 读入优化 << 关同步 cin << 此优化
// 用 isdigit() 会小幅变慢
// 返回 false 表示读到文件尾
namespace Reader {
    const int L = (1 << 15) + 5;
    char buffer[L], *S, *T;
    __inline bool getchar(char &ch) {
        if (S == T) {
            T = (S = buffer) + fread(buffer, 1, L, stdin);
            if (S == T) {
                ch = EOF;
                return false;
            }
        }
        ch = *S++;
        return true;
    }
    __inline bool getint(int &x) {
        char ch; bool neg = 0;
        for (; getchar(ch) && (ch < '0' || ch > '9'); ) neg ^= ch == '-';
        if (ch == EOF) return false;
        x = ch - '0';
        for (; getchar(ch), ch >= '0' && ch <= '9'; )
            x = x * 10 + ch - '0';
        if (neg) x = -x;
        return true;
    }
}
```

真正释放 STL 内存

```
template <typename T>
__inline void clear(T& container) {
    container.clear(); // 或者删除了一堆元素
    T(container).swap(container);
}
```

梅森旋转算法

```
template <typename T>
__inline void clear(T& container) {
    container.clear(); // 或者删除了一堆元素
    T(container).swap(container);
}
```

蔡勒公式

```
int solve(int year, int month, int day) {
    int answer;
    if (month == 1 || month == 2) {
        month += 12;
        year--;
    }
    if ((year < 1752) || (year == 1752 && month < 9) ||
        (year == 1752 && month == 9 && day < 3)) {
        answer = (day + 2 * month + 3 * (month + 1) / 5 + year + year / 4 + 5) % 7;
    } else {
        answer = (day + 2 * month + 3 * (month + 1) / 5 + year + year / 4
               - year / 100 + year / 400) % 7;
    }
    return answer;
}
```

开栈

```
register char *_sp __asm__("rsp");
int main() {
    const int size = 400 << 20;//400MB
    static char *sys, *mine(new char[size] + size - 4096);
    sys = _sp; _sp = mine; _main(); _sp = sys;
}
```

Size 为 k 的子集

```
void solve(int n, int k) {
    for (int comb = (1 << k) - 1; comb < (1 << n); ) {
        // ...
        int x = comb & -comb, y = comb + x;
        comb = (((comb & ~y) / x) >> 1) | y;
    }
}
```

长方体表面两点最短距离

```
int r;
void turn(int i, int j, int x, int y, int z,int x0, int y0, int L, int W, int H) {
    if (z==0) { int R = x*x+y*y; if (R<r) r=R;
    } else {
        if(i>=0 && i< 2) turn(i+1, j, x0+L+z, y, x0+L-x, x0+L, y0, H, W, L);
        if(j>=0 && j< 2) turn(i, j+1, x, y0+W+z, y0+W-y, x0, y0+W, L, H, W);
        if(i<=0 && i>-2) turn(i-1, j, x0-z, y, x-x0, x0-H, y0, H, W, L);
        if(j<=0 && j>-2) turn(i, j-1, x, y0-z, y-y0, x0, y0-H, L, H, W);
    }
}
int main(){
    int L, H, W, x1, y1, z1, x2, y2, z2;
    cin >> L >> W >> H >> x1 >> y1 >> z1 >> x2 >> y2 >> z2;
    if (z1!=0 && z1!=H) if (y1==0 || y1==W)
         swap(y1,z1), std::swap(y2,z2), std::swap(W,H);
    else swap(x1,z1), std::swap(x2,z2), std::swap(L,H);
    if (z1==H) z1=0, z2=H-z2;
    r=0x3fffffff;
    turn(0,0,x2-x1,y2-y1,z2,-x1,-y1,L,W,H);
    cout<<r<<endl;
}
```

经纬度求球面最短距离

```
double sphereDis(double lon1, double lat1, double lon2, double lat2, double R) {
    return R * acos(cos(lat1) * cos(lat2) * cos(lon1 - lon2) + sin(lat1) * sin(lat2));
}
```

32-bit/64-bit 随机素数

32-bit	64-bit
73550053	1249292846855685773
148898719	1701750434419805569
189560747	3605499878424114901
459874703	5648316673387803781
1202316001	6125342570814357977
1431183547	6215155308775851301
1438011109	6294606778040623451
1538762023	6347330550446020547
1557944263	7429632924303725207
1981315913	8524720079480389849

NTT 素数及其原根

Prime	Primitive root
1053818881	7
1051721729	6
1045430273	3
1012924417	5
1007681537	3

Arithmetic Function

$$\sigma_k(n) = \sum_{d|n} d^k = \prod_{i=1}^{\omega(n)} \frac{p_i^{(a_i+1)k} - 1}{p_i^k - 1}$$

$$J_k(n) = n^k \prod_{p|n} (1 - \frac{1}{p^k})$$

$J_k(n)$ is the number of k-tuples of positive integers all less than or equal to n that form a coprime $(k+1)$-tuple together with n.

$$\sum_{\delta|n} J_k(\delta) = n^k$$

$$\sum_{\delta|n} \delta^s J_r(\delta) J_s(\frac{n}{\delta}) = J_{r+s}(n)$$

$$\sum_{\delta|n} \varphi(\delta) d(\frac{n}{\delta}) = \sigma(n),\ \sum_{\delta|n} |\mu(\delta)| = 2^{\omega(n)}$$

$$\sum_{\delta|n} 2^{\omega(\delta)} = d(n^2),\ \sum_{\delta|n} d(\delta^2) = d^2(n)$$

$$\sum_{\delta|n} d(\frac{n}{\delta}) 2^{\omega(\delta)} = d^2(n),\ \sum_{\delta|n} \frac{\mu(\delta)}{\delta} = \frac{\varphi(n)}{n}$$

$$\sum_{\delta|n} \frac{\mu(\delta)}{\varphi(\delta)} = d(n),\ \sum_{\delta|n} \frac{\mu^2(\delta)}{\varphi(\delta)} = \frac{n}{\varphi(n)}$$

$$n|\varphi(a^n-1)$$

$$\sum_{\substack{1\le k\le n\\ \gcd(k,n)=1}} f(\gcd(k-1,n)) = \varphi(n)\sum_{d|n}\frac{(\mu*f)(d)}{\varphi(d)}$$

$$\varphi(\mathrm{lcm}(m,n))\varphi(\gcd(m,n)) = \varphi(m)\varphi(n)$$

$$\sum_{\delta|n} d^3(\delta) = (\sum_{\delta|n} d(\delta))^2$$

$$d(uv) = \sum_{\delta|\gcd(u,v)} \mu(\delta)d(\frac{u}{\delta})d(\frac{v}{\delta})$$

$$\sigma_k(u)\sigma_k(v) = \sum_{\delta|\gcd(u,v)} \delta^k\sigma_k(\frac{uv}{\delta^2})$$

$$\mu(n) = \sum_{k=1}^{n}[\gcd(k,n)=1]\cos 2\pi\frac{k}{n}$$

$$\varphi(n) = \sum_{k=1}^{n}[\gcd(k,n)=1] = \sum_{k=1}^{n}\gcd(k,n)\cos 2\pi\frac{k}{n}$$

$$\begin{cases} S(n) = \sum_{k=1}^{n}(f*g)(k) \\ \sum_{k=1}^{n} S(\lfloor\frac{n}{k}\rfloor) = \sum_{i=1}^{n} f(i)\sum_{j=1}^{\lfloor\frac{n}{i}\rfloor}(g*1)(j) \end{cases}$$

$$\begin{cases} S(n) = \sum_{k=1}^{n}(f\cdot g)(k), g \text{ completely multiplicative} \\ \sum_{k=1}^{n} S(\lfloor\frac{n}{k}\rfloor)g(k) = \sum_{k=1}^{n}(f*1)(k)g(k) \end{cases}$$

Binomial Coefficients

$$\binom{n}{k} = (-1)^k\binom{k-n-1}{k}$$

$$\sum_{k\le n}\binom{r+k}{k} = \binom{r+n+1}{n}$$

$$\sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$$

$$\sqrt{1+z} = 1+\sum_{k=1}^{\infty}\frac{(-1)^{k-1}}{k\times 2^{2k-1}}\binom{2k-2}{k-1}z^k$$

$$\sum_{k=0}^{r}\binom{r-k}{m}\binom{s+k}{n} = \binom{r+s+1}{m+n+1}$$

$$C_{n,m} = \binom{n+m}{m} - \binom{n+m}{m-1}, n\ge m$$

$$\binom{n}{k} \equiv [n\&k = k] \pmod 2$$

Fibonacci Numbers

$$F(z) = \frac{z}{1 - z - z^2}$$

$$f_n = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}}, \phi = \frac{1+\sqrt{5}}{2}, \hat{\phi} = \frac{1-\sqrt{5}}{2}$$

$$\sum_{k=1}^{n} f_k = f_{n+2} - 1$$

$$\sum_{k=1}^{n} f_k^2 = f_n f_{n+1}$$

$$\sum_{k=0}^{n} f_k f_{n-k} = \frac{1}{5}(n-1)f_n + \frac{2}{5} n f_{n-1}$$

$$f_n^2 + (-1)^n = f_{n+1} f_{n-1}$$

$$f_{n+k} = f_n f_{k+1} + f_{n-1} f_k$$

$$f_{2n+1} = f_n^2 + f_{n+1}^2$$

$$(-1)^k f_{n-k} = f_n f_{k-1} - f_{n-1} f_k$$

$$\text{Modulo } f_n, f_{mn+r} \equiv \begin{cases} f_r, & m \bmod 4 = 0; \\ (-1)^{r+1} f_{n-r}, & m \bmod 4 = 1; \\ (-1)^n f_r, & m \bmod 4 = 2; \\ (-1)^{r+1+n} f_{n-r}, & m \bmod 4 = 3. \end{cases}$$

Stirling Cycle Numbers

$$\left[{n+1 \atop k}\right] = n\left[{n \atop k}\right] + \left[{n \atop k-1}\right], \quad \left[{n+1 \atop 2}\right] = n! H_n$$

$$x^{\underline{n}} = \sum_k \left[{n \atop k}\right] (-1)^{n-k} x^k, \quad x^{\overline{n}} = \sum_k \left[{n \atop k}\right] x^k$$

Stirling Subset Numbers

$$\left\{{n+1 \atop k}\right\} = k\left\{{n \atop k}\right\} + \left\{{n \atop k-1}\right\}$$

$$x^n = \sum_k \left\{{n \atop k}\right\} x^{\underline{k}} = \sum_k \left\{{n \atop k}\right\} (-1)^{n-k} x^{\overline{k}}$$

$$m! \left\{{n \atop m}\right\} = \sum_k \binom{m}{k} k^n (-1)^{m-k}$$

Eulerian Numbers

$$\left\langle {n \atop k} \right\rangle = (k+1)\left\langle {n-1 \atop k} \right\rangle + (n-k)\left\langle {n-1 \atop k-1} \right\rangle$$

$$x^n = \sum_k \left\langle {n \atop k} \right\rangle \binom{x+k}{n}$$

$$\left\langle {n \atop m} \right\rangle = \sum_{k=0}^{m} \binom{n+1}{k} (m+1-k)^n (-1)^k$$

Harmonic Numbers

$$\sum_{k=1}^{n} H_k = (n+1)H_n - n$$

$$\sum_{k=1}^{n} kH_k = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}$$

$$\sum_{k=1}^{n} \binom{k}{m} H_k = \binom{n+1}{m+1}(H_{n+1} - \frac{1}{m+1})$$

Pentagonal Number Theorem

$$\prod_{n=1}^{\infty}(1-x^n) = \sum_{n=-\infty}^{\infty} (-1)^k x^{k(3k-1)/2}$$

$$p(n) = p(n-1) + p(n-2) - p(n-5) - p(n-7) + \cdots$$

$$f(n,k) = p(n) - p(n-k) - p(n-2k) + p(n-5k) + p(n-7k) - \cdots$$

Bell Numbers

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$$

$$B_{p^m+n} \equiv mB_n + B_{n+1} \pmod{p}$$

Bernoulli Numbers

$$B_n = 1 - \sum_{k=0}^{n-1} \binom{n}{k} \frac{B_k}{n-k+1}$$

$$G(x) = \sum_{k=0}^{\infty} \frac{B_k}{k!} x^k = \frac{1}{\sum_{k=0}^{\infty} \frac{x^k}{(k+1)!}}$$

$$S_m(n) = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_k n^{m-k+1}$$

Tetrahedron Volume

$$V = \frac{\sqrt{4u^2v^2w^2 - \sum_{cyc} u^2(v^2+w^2-U^2)^2 + \prod_{cyc}(v^2+w^2-U^2)}}{12}$$

BEST Thoerem

Counting the number of different Eulerian circuits in directed graphs.

$$\mathrm{ec}(G) = t_w(G)\prod_{v\in V}(\deg(v)-1)!$$

When calculating $t_w(G)$ for directed multigraphs, the entry $q_{i,j}$ for distinct i and j equals $-m$, where m is the number of edges from i to j, and the entry $q_{i,i}$ equals the indegree of i minus the number of loops at i. It is a property of Eulerian graphs that $\mathrm{tv}(G) = \mathrm{tw}(G)$ for every two vertices v and w in a connected Eulerian graph G.

重心

半径为 r , 圆心角为 θ 的扇形重心与圆心的距离为 $\frac{4r\sin(\theta/2)}{3\theta}$
半径为 r , 圆心角为 θ 的圆弧重心与圆心的距离为 $\frac{4r\sin^3(\theta/2)}{3(\theta-\sin(\theta))}$

Others

$$S_j = \sum_{k=1}^{n} x_k^j$$

$$h_m = \sum_{1\le j_1<\cdots<j_m\le n} x_{j_1}\cdots x_{j_m}$$

$$H_m = \sum_{1\le j_1\le\cdots\le j_m\le n} x_{j_1}\cdots x_{j_m}$$

$$h_n = \frac{1}{n}\sum_{k=1}^{n}(-1)^{k+1}S_k h_{n-k}$$

$$H_n = \frac{1}{n}\sum_{k=1}^{n}S_k H_{n-k}$$

$$\sum_{k=0}^{n} kc^k = \frac{nc^{n+2}-(n+1)c^{n+1}+c}{(c-1)^2}$$

$$n! = \sqrt{2\pi n}(\frac{n}{e})^n(1+\frac{1}{12n}+\frac{1}{288n^2}+O(\frac{1}{n^3}))$$

$$\begin{aligned}&\max\{x_a-x_b, y_a-y_b, z_a-z_b\} - \min\{x_a-x_b, y_a-y_b, z_a-z_b\}\\ =&\frac{1}{2}\sum_{cyc}|(x_a-y_a)-(x_b-y_b)|\end{aligned}$$

$$(a+b)(b+c)(c+a) = \frac{(a+b+c)^3-a^3-b^3-c^3}{3}$$

Formulas

Integrals of Rational Functions

$$\int \frac{1}{1+x^2} dx = \tan^{-1} x \qquad (1)$$

$$\int \frac{1}{a^2+x^2} dx = \frac{1}{a} \tan^{-1} \frac{x}{a} \qquad (2)$$

$$\int \frac{x}{a^2+x^2} dx = \frac{1}{2} \ln |a^2+x^2| \qquad (3)$$

$$\int \frac{x^2}{a^2+x^2} dx = x - a \tan^{-1} \frac{x}{a} \qquad (4)$$

$$\int \frac{x^3}{a^2+x^2} dx = \frac{1}{2}x^2 - \frac{1}{2}a^2 \ln |a^2+x^2| \qquad (5)$$

$$\int \frac{1}{ax^2+bx+c} dx = \frac{2}{\sqrt{4ac-b^2}} \tan^{-1} \frac{2ax+b}{\sqrt{4ac-b^2}} \qquad (6)$$

$$\int \frac{1}{(x+a)(x+b)} dx = \frac{1}{b-a} \ln \frac{a+x}{b+x},\ a \neq b \qquad (7)$$

$$\int \frac{x}{(x+a)^2} dx = \frac{a}{a+x} + \ln |a+x| \qquad (8)$$

$$\int \frac{x}{ax^2+bx+c} dx = \frac{1}{2a} \ln |ax^2+bx+c| - \frac{b}{a\sqrt{4ac-b^2}} \tan^{-1} \frac{2ax+b}{\sqrt{4ac-b^2}} \qquad (9)$$

Integrals with Roots

$$\int \frac{x}{\sqrt{x \pm a}} dx = \frac{2}{3}(x \mp 2a)\sqrt{x \pm a} \qquad (10)$$

$$\int \sqrt{\frac{x}{a-x}} dx = -\sqrt{x(a-x)} - a \tan^{-1} \frac{\sqrt{x(a-x)}}{x-a} \qquad (11)$$

$$\int \sqrt{\frac{x}{a+x}} dx = \sqrt{x(a+x)} - a \ln \left[\sqrt{x} + \sqrt{x+a}\right] \qquad (12)$$

$$\int x\sqrt{ax+b} dx = \frac{2}{15a^2}(-2b^2+abx+3a^2x^2)\sqrt{ax+b} \qquad (13)$$

$$\int \sqrt{x(ax+b)} dx = \frac{1}{4a^{3/2}} \left[(2ax+b)\sqrt{ax(ax+b)} - b^2 \ln \left| a\sqrt{x} + \sqrt{a(ax+b)} \right| \right] \qquad (14)$$

$$\int \sqrt{x^3(ax+b)} dx = \left[\frac{b}{12a} - \frac{b^2}{8a^2x} + \frac{x}{3}\right] \sqrt{x^3(ax+b)} + \frac{b^3}{\,} \ln \left| a\sqrt{x} + \sqrt{a(ax+b)} \right| \qquad (15)$$

$$\int \sqrt{x^2 \pm a^2} dx = \frac{1}{2}x\sqrt{x^2 \pm a^2} \pm \frac{1}{2}a^2 \ln \left| x + \sqrt{x^2 \pm a^2} \right| \qquad (16)$$

$$\int \sqrt{a^2 - x^2} dx = \frac{1}{2}x\sqrt{a^2-x^2} + \frac{1}{2}a^2 \tan^{-1} \frac{x}{\sqrt{a^2-x^2}} \qquad (17)$$

$$\int x\sqrt{x^2 \pm a^2} dx = \frac{1}{3}\left(x^2 \pm a^2\right)^{3/2} \qquad (18)$$

$$\int \frac{1}{\sqrt{x^2 \pm a^2}} dx = \ln \left| x + \sqrt{x^2 \pm a^2} \right| \qquad (19)$$

$$\int \frac{1}{\sqrt{a^2 - x^2}} dx = \sin^{-1} \frac{x}{a} \qquad (20)$$

$$\int \frac{x}{\sqrt{x^2 \pm a^2}} dx = \sqrt{x^2 \pm a^2} \qquad (21)$$

$$\int \frac{x}{\sqrt{a^2 - x^2}} dx = -\sqrt{a^2 - x^2} \qquad (22)$$

$$\int \frac{x^2}{\sqrt{x^2 \pm a^2}} dx = \frac{1}{2}x\sqrt{x^2 \pm a^2} \mp \frac{1}{2}a^2 \ln \left| x + \sqrt{x^2 \pm a^2} \right| \qquad (23)$$

$$\int \sqrt{ax^2+bx+c} dx = \frac{b+2ax}{4a}\sqrt{ax^2+bx+c} + \frac{4ac-b^2}{8a^{3/2}} \ln \left| 2ax + b + 2\sqrt{a(ax^2+bx^{+}c)} \right| \qquad (24)$$

$$\int x\sqrt{ax^2+bx+c} = \frac{1}{48a^{5/2}} \left(2\sqrt{a}\sqrt{ax^2+bx+c} \times \left(-3b^2 + 2abx + 8a(c+ax^2)\right) + 3(b^3-4abc) \ln \left| b + 2ax + 2\sqrt{a}\sqrt{ax^2+bx+c} \right| \right) \qquad (25)$$

$$\int \frac{1}{\sqrt{ax^2+bx+c}} dx = \frac{1}{\sqrt{a}} \ln \left| 2ax + b + 2\sqrt{a(ax^2+bx+c)} \right| \qquad (26)$$

$$\int \frac{x}{\sqrt{ax^2+bx+c}} dx = \frac{1}{a}\sqrt{ax^2+bx+c} - \frac{b}{2a^{3/2}} \ln \left| 2ax + b + 2\sqrt{a(ax^2+bx+c)} \right| \qquad (27)$$

$$\int \frac{dx}{(a^2+x^2)^{3/2}} = \frac{x}{a^2\sqrt{a^2+x^2}} \qquad (28)$$

Integrals with Logarithms

$$\int \frac{\ln ax}{x} dx = \frac{1}{2} (\ln ax)^2 \qquad (29)$$

$$\int \ln(ax+b) dx = \left(x + \frac{b}{a}\right) \ln(ax+b) - x, a \neq 0 \qquad (30)$$

$$\int \ln(x^2+a^2)\, \mathrm{dx} = x \ln(x^2+a^2) + 2a \tan^{-1} \frac{x}{a} - 2x \qquad (31)$$

$$\int \ln(x^2-a^2)\, \mathrm{dx} = x \ln(x^2-a^2) + a \ln \frac{x+a}{x-a} - 2x \qquad (32)$$

$$\int \ln\left(ax^2+bx+c\right) dx = \frac{1}{a}\sqrt{4ac-b^2} \tan^{-1} \frac{2ax+b}{\sqrt{4ac-b^2}} - 2x + \left(\frac{b}{2a} + x\right) \ln\left(ax^2+bx+c\right) \qquad (33)$$

$$\int x \ln(ax+b) dx = \frac{bx}{2a} - \frac{1}{4}x^2 + \frac{1}{2}\left(x^2 - \frac{b^2}{a^2}\right) \ln(ax+b) \qquad (34)$$

$$\int x \ln\left(a^2 - b^2x^2\right) dx = -\frac{1}{2}x^2 + \frac{1}{2}\left(x^2 - \frac{a^2}{b^2}\right) \ln\left(a^2 - b^2x^2\right) \qquad (35)$$

Integrals with Exponentials

$$\int x^n e^{ax}\, \mathrm{dx} = \frac{x^n e^{ax}}{a} - \frac{n}{a} \int x^{n-1} e^{ax}\, \mathrm{dx} \qquad (36)$$

$$\int x e^{-ax^2}\, \mathrm{dx} = -\frac{1}{2a} e^{-ax^2} \qquad (37)$$

Integrals with Trigonometric Functions

$$\int \sin^3 ax dx = -\frac{3\cos ax}{4a} + \frac{\cos 3ax}{12a} \qquad (38)$$

$$\int \cos^2 ax dx = \frac{x}{2} + \frac{\sin 2ax}{4a} \qquad (39)$$

$$\int \cos^3 ax dx = \frac{3\sin ax}{4a} + \frac{\sin 3ax}{12a} \qquad (40)$$

$$\int \cos ax \sin bx dx = \frac{\cos[(a-b)x]}{2(a-b)} - \frac{\cos[(a+b)x]}{2(a+b)}, a \neq b \qquad (41)$$

$$\int \sin^2 ax \cos bx dx = -\frac{\sin[(2a-b)x]}{4(2a-b)} + \frac{\sin bx}{2b} - \frac{\sin[(2a+b)x]}{4(2a+b)} \qquad (42)$$

$$\int \sin^2 x \cos x dx = \frac{1}{\,} \sin^3 x \qquad (43)$$

$$\int \cos^2 ax \sin bx dx = \frac{\cos[(2a-b)x]}{4(2a-b)} - \frac{\cos bx}{2b} - \frac{\cos[(2a+b)x]}{4(2a+b)} \tag{44}$$

$$\int \cos^2 ax \sin ax dx = -\frac{1}{3a}\cos^3 ax \tag{45}$$

$$\int \sin^2 ax \cos^2 bx dx = \frac{x}{4} - \frac{\sin 2ax}{8a} - \frac{\sin[2(a-b)x]}{16(a-b)} + \frac{\sin 2bx}{8b} - \frac{\sin[2(a+b)x]}{16(a+b)} \tag{46}$$

$$\int \sin^2 ax \cos^2 ax dx = \frac{x}{8} - \frac{\sin 4ax}{32a} \tag{47}$$

$$\int \tan ax dx = -\frac{1}{a}\ln\cos ax \tag{48}$$

$$\int \tan^2 ax dx = -x + \frac{1}{a}\tan ax \tag{49}$$

$$\int \tan^3 ax dx = \frac{1}{a}\ln\cos ax + \frac{1}{2a}\sec^2 ax \tag{50}$$

$$\int \sec x dx = \ln|\sec x + \tan x| = 2\tanh^{-1}\left(\tan\frac{x}{2}\right) \tag{51}$$

$$\int \sec^2 ax dx = \frac{1}{a}\tan ax \tag{52}$$

$$\int \sec^3 x \,\mathrm{dx} = \frac{1}{2}\sec x\tan x + \frac{1}{2}\ln|\sec x + \tan x| \tag{53}$$

$$\int \sec x \tan x dx = \sec x \tag{54}$$

$$\int \sec^2 x \tan x dx = \frac{1}{2}\sec^2 x \tag{55}$$

$$\int \sec^n x \tan x dx = \frac{1}{n}\sec^n x, n \neq 0 \tag{56}$$

$$\int \csc x dx = \ln\left|\tan\frac{x}{2}\right| = \ln|\csc x - \cot x| + C \tag{57}$$

$$\int \csc^2 ax dx = -\frac{1}{a}\cot ax \tag{58}$$

$$\int \csc^3 x dx = -\frac{1}{2}\cot x \csc x + \frac{1}{2}\ln|\csc x - \cot x| \tag{59}$$

$$\int \csc^n x \cot x dx = -\frac{1}{n}\csc^n x, n \neq 0 \tag{60}$$

$$\int \sec x \csc x dx = \ln|\tan x| \tag{61}$$

Products of Trigonometric Functions and Monomials

$$\int x\cos x dx = \cos x + x\sin x \tag{62}$$

$$\int x\cos ax dx = \frac{1}{a^2}\cos ax + \frac{x}{a}\sin ax \tag{63}$$

$$\int x^2\cos x dx = 2x\cos x + \left(x^2 - 2\right)\sin x \tag{64}$$

$$\int x^2\cos ax dx = \frac{2x\cos ax}{a^2} + \frac{a^2x^2 - 2}{a^3}\sin ax \tag{65}$$

$$\int x\sin x dx = -x\cos x + \sin x \tag{66}$$

$$\int x\sin ax dx = -\frac{x\cos ax}{a} + \frac{\sin ax}{a^2} \tag{67}$$

$$\int x^2\sin x dx = \left(2 - x^2\right)\cos x + 2x\sin x \tag{68}$$

$$\int x^2\sin ax dx = \frac{2 - a^2x^2}{a^3}\cos ax + \frac{2x\sin ax}{a^2} \tag{69}$$

Products of Trigonometric Functions and Exponentials

$$\int e^x\sin x dx = \frac{1}{2}e^x(\sin x - \cos x) \tag{70}$$

$$\int e^{bx}\sin ax dx = \frac{1}{a^2+b^2}e^{bx}(b\sin ax - a\cos ax) \tag{71}$$

$$\int e^x\cos x dx = \frac{1}{2}e^x(\sin x + \cos x) \tag{72}$$

$$\int e^{bx}\cos ax dx = \frac{1}{a^2+b^2}e^{bx}(a\sin ax + b\cos ax) \tag{73}$$

$$\int xe^x\sin x dx = \frac{1}{2}e^x(\cos x - x\cos x + x\sin x) \tag{74}$$

$$\int xe^x\cos x dx = \frac{1}{2}e^x(x\cos x - \sin x + x\sin x) \tag{75}$$

Java

```
import java.io.*;
import java.util.*;
import java.math.*;
public class Main {
    public static void main(String[] args) {
        InputStream inputStream = System.in;
        OutputStream outputStream = System.out;
        InputReader in = new InputReader(inputStream);
        PrintWriter out = new PrintWriter(outputStream);
    }
}
public static class edge implements Comparable<edge>{
    public int u,v,w;
    public int compareTo(edge e){
        return w-e.w;
    }
}
public static class cmp implements Comparator<edge>{
    public int compare(edge a,edge b){
        if(a.w<b.w)return 1;
        if(a.w>b.w)return -1;
        return 0;
    }
}
class InputReader {
    public BufferedReader reader;
    public StringTokenizer tokenizer;

    public InputReader(InputStream stream) {
        reader = new BufferedReader(new InputStreamReader(stream), 32768);
        tokenizer = null;
    }

    public String next() {
        while (tokenizer == null || !tokenizer.hasMoreTokens()) {
            try {
                tokenizer = new StringTokenizer(reader.readLine());
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
        return tokenizer.nextToken();
    }

```

```java
    public int nextInt() {
        return Integer.parseInt(next());
    }

    public long nextLong() {
        return Long.parseLong(next());
    }
}
```

compact1, compact2, compact3

java.math

Class BigInteger

java.lang.Object
    java.lang.Number
        java.math.BigInteger

All Implemented Interfaces:

`Serializable, Comparable<BigInteger>`

```
public class BigInteger
extends Number
implements Comparable<BigInteger>
```

Immutable arbitrary-precision integers. All operations behave as if BigIntegers were represented in two's-complement notation (like Java's primitive integer types). BigInteger provides analogues to all of Java's primitive integer operators, and all relevant methods from java.lang.Math. Additionally, BigInteger provides operations for modular arithmetic, GCD calculation, primality testing, prime generation, bit manipulation, and a few other miscellaneous operations.

Semantics of arithmetic operations exactly mimic those of Java's integer arithmetic operators, as defined in *The Java Language Specification*. For example, division by zero throws an `ArithmeticException`, and division of a negative by a positive yields a negative (or zero) remainder. All of the details in the Spec concerning overflow are ignored, as BigIntegers are made as large as necessary to accommodate the results of an operation.

Semantics of shift operations extend those of Java's shift operators to allow for negative shift distances. A right-shift with a negative shift distance results in a left shift, and vice-versa. The unsigned right shift operator (>>>) is omitted, as this operation makes little sense in combination with the "infinite word size" abstraction provided by this class.

Semantics of bitwise logical operations exactly mimic those of Java's bitwise integer operators. The binary operators (`and`, `or`, `xor`) implicitly perform sign extension on the shorter of the two operands prior to performing the operation.

Comparison operations perform signed integer comparisons, analogous to those performed by Java's relational and equality operators.

Modular arithmetic operations are provided to compute residues, perform exponentiation, and compute multiplicative inverses. These methods always return a non-negative result, between `0` and `(modulus - 1)`, inclusive.

Bit operations operate on a single bit of the two's-complement representation of their operand. If necessary, the operand is sign- extended so that it contains the designated bit. None of the single-bit operations can produce a BigInteger with a different sign from the BigInteger being operated on, as they affect only a single bit, and the "infinite word size" abstraction provided by this class ensures that there are infinitely many "virtual sign bits" preceding each BigInteger.

For the sake of brevity and clarity, pseudo-code is used throughout the descriptions of BigInteger methods. The pseudo-code expression `(i + j)` is shorthand for "a BigInteger whose value is that of the BigInteger `i` plus that of the BigInteger `j`." The pseudo-code expression `(i == j)` is shorthand for "`true` if and only if the BigInteger `i` represents the same value as the BigInteger `j`." Other pseudo-code expressions are interpreted similarly.

All methods and constructors in this class throw `NullPointerException` when passed a null object reference for any input parameter. BigInteger must support values in the range $-2^{\texttt{Integer.MAX\_VALUE}}$ (exclusive) to $+2^{\texttt{Integer.MAX\_VALUE}}$ (exclusive) and may support values outside of that range. The range of probable prime values is limited and may be less than the full supported positive range of `BigInteger`. The range must be at least 1 to $2^{500000000}$.

Implementation Note:

BigInteger constructors and operations throw ArithmeticException when the result is out of the supported range of $-2^{\texttt{Integer.MAX\_VALUE}}$ (exclusive) to $+2^{\texttt{Integer.MAX\_VALUE}}$ (exclusive).

Since:

JDK1.1

See Also:

BigDecimal, Serialized Form

Field Summary

Fields

Modifier and Type	Field and Description
static **BigInteger**	**ONE** The BigInteger constant one.
static **BigInteger**	**TEN** The BigInteger constant ten.
static **BigInteger**	**ZERO** The BigInteger constant zero.

Constructor Summary

Constructors

Constructor and Description
BigInteger(byte[] val) Translates a byte array containing the two's-complement binary representation of a BigInteger into a BigInteger.
BigInteger(int signum, byte[] magnitude) Translates the sign-magnitude representation of a BigInteger into a BigInteger.

BigInteger(int bitLength, int certainty, **Random** rnd)
Constructs a randomly generated positive BigInteger that is probably prime, with the specified bitLength.

BigInteger(int numBits, **Random** rnd)
Constructs a randomly generated BigInteger, uniformly distributed over the range 0 to ($2^{numBits}$ - 1), inclusive.

BigInteger(**String** val)
Translates the decimal String representation of a BigInteger into a BigInteger.

BigInteger(**String** val, int radix)
Translates the String representation of a BigInteger in the specified radix into a BigInteger.

Method Summary

All Methods Static Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
BigInteger	**abs**() Returns a BigInteger whose value is the absolute value of this BigInteger.
BigInteger	**add**(**BigInteger** val) Returns a BigInteger whose value is `(this + val)`.
BigInteger	**and**(**BigInteger** val) Returns a BigInteger whose value is `(this & val)`.
BigInteger	**andNot**(**BigInteger** val) Returns a BigInteger whose value is `(this & ~val)`.
int	**bitCount**() Returns the number of bits in the two's complement representation of this BigInteger that differ from its sign bit.
int	**bitLength**() Returns the number of bits in the minimal two's-complement representation of this BigInteger, *excluding* a sign bit.
byte	**byteValueExact**() Converts this `BigInteger` to a `byte`, checking for lost information.
BigInteger	**clearBit**(int n) Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit cleared.
int	**compareTo**(**BigInteger** val) Compares this BigInteger with the specified BigInteger.
BigInteger	**divide**(**BigInteger** val) Returns a BigInteger whose value is `(this / val)`.

	Returns a BigInteger whose value is `(this / val)`.
BigInteger[]	**divideAndRemainder**(**BigInteger** val) Returns an array of two BigIntegers containing `(this / val)` followed by `(this % val)`.
double	**doubleValue**() Converts this BigInteger to a `double`.
boolean	**equals**(**Object** x) Compares this BigInteger with the specified Object for equality.
BigInteger	**flipBit**(int n) Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit flipped.
float	**floatValue**() Converts this BigInteger to a `float`.
BigInteger	**gcd**(**BigInteger** val) Returns a BigInteger whose value is the greatest common divisor of `abs(this)` and `abs(val)`.
int	**getLowestSetBit**() Returns the index of the rightmost (lowest-order) one bit in this BigInteger (the number of zero bits to the right of the rightmost one bit).
int	**hashCode**() Returns the hash code for this BigInteger.
int	**intValue**() Converts this BigInteger to an `int`.
int	**intValueExact**() Converts this `BigInteger` to an `int`, checking for lost information.
boolean	**isProbablePrime**(int certainty) Returns `true` if this BigInteger is probably prime, `false` if it's definitely composite.
long	**longValue**() Converts this BigInteger to a `long`.
long	**longValueExact**() Converts this `BigInteger` to a `long`, checking for lost information.
BigInteger	**max**(**BigInteger** val) Returns the maximum of this BigInteger and `val`.
BigInteger	**min**(**BigInteger** val) Returns the minimum of this BigInteger and `val`.
BigInteger	**mod**(**BigInteger** m) Returns a BigInteger whose value is `(this mod m)`.

BigInteger	**modInverse**(BigInteger m) Returns a BigInteger whose value is (this^{-1} mod m).
BigInteger	**modPow**(BigInteger exponent, BigInteger m) Returns a BigInteger whose value is (this^{exponent} mod m).
BigInteger	**multiply**(BigInteger val) Returns a BigInteger whose value is (this * val).
BigInteger	**negate**() Returns a BigInteger whose value is (-this).
BigInteger	**nextProbablePrime**() Returns the first integer greater than this BigInteger that is probably prime.
BigInteger	**not**() Returns a BigInteger whose value is (~this).
BigInteger	**or**(BigInteger val) Returns a BigInteger whose value is (this | val).
BigInteger	**pow**(int exponent) Returns a BigInteger whose value is (this^{exponent}).
static BigInteger	**probablePrime**(int bitLength, Random rnd) Returns a positive BigInteger that is probably prime, with the specified bitLength.
BigInteger	**remainder**(BigInteger val) Returns a BigInteger whose value is (this % val).
BigInteger	**setBit**(int n) Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit set.
BigInteger	**shiftLeft**(int n) Returns a BigInteger whose value is (this << n).
BigInteger	**shiftRight**(int n) Returns a BigInteger whose value is (this >> n).
short	**shortValueExact**() Converts this BigInteger to a short, checking for lost information.
int	**signum**() Returns the signum function of this BigInteger.
BigInteger	**subtract**(BigInteger val) Returns a BigInteger whose value is (this - val).
boolean	**testBit**(int n) Returns true if and only if the designated bit is set.
byte[]	**toByteArray**()

	Returns a byte array containing the two's-complement representation of this BigInteger.
String	toString() Returns the decimal String representation of this BigInteger.
String	toString(int radix) Returns the String representation of this BigInteger in the given radix.
static BigInteger	valueOf(long val) Returns a BigInteger whose value is equal to that of the specified long.
BigInteger	xor(BigInteger val) Returns a BigInteger whose value is (this ^ val).

Methods inherited from class java.lang.Number

byteValue, shortValue

Methods inherited from class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Field Detail

ZERO

`public static final BigInteger ZERO`

The BigInteger constant zero.

Since:

1.2

ONE

`public static final BigInteger ONE`

The BigInteger constant one.

Since:

1.2

TEN

`public static final BigInteger TEN`

The BigInteger constant ten.

compact1, compact2, compact3

java.util

Class TreeMap<K,V>

java.lang.Object
    java.util.AbstractMap<K,V>
        java.util.TreeMap<K,V>

Type Parameters:

`K` - the type of keys maintained by this map

`V` - the type of mapped values

All Implemented Interfaces:

`Serializable, Cloneable, Map<K,V>, NavigableMap<K,V>, SortedMap<K,V>`

```
public class TreeMap<K,V>
extends AbstractMap<K,V>
implements NavigableMap<K,V>, Cloneable, Serializable
```

A Red-Black tree based `NavigableMap` implementation. The map is sorted according to the natural ordering of its keys, or by a `Comparator` provided at map creation time, depending on which constructor is used.

This implementation provides guaranteed log(n) time cost for the `containsKey`, `get`, `put` and `remove` operations. Algorithms are adaptations of those in Cormen, Leiserson, and Rivest's *Introduction to Algorithms*.

Note that the ordering maintained by a tree map, like any sorted map, and whether or not an explicit comparator is provided, must be *consistent with `equals`* if this sorted map is to correctly implement the `Map` interface. (See `Comparable` or `Comparator` for a precise definition of *consistent with equals*.) This is so because the `Map` interface is defined in terms of the `equals` operation, but a sorted map performs all key comparisons using its `compareTo` (or `compare`) method, so two keys that are deemed equal by this method are, from the standpoint of the sorted map, equal. The behavior of a sorted map *is* well-defined even if its ordering is inconsistent with `equals`; it just fails to obey the general contract of the `Map` interface.

Note that this implementation is not synchronized. If multiple threads access a map concurrently, and at least one of the threads modifies the map structurally, it *must* be synchronized externally. (A structural modification is any operation that adds or deletes one or more mappings; merely changing the value associated with an existing key is not a structural modification.) This is typically accomplished by synchronizing on some object that naturally encapsulates the map. If no such object exists, the map should be "wrapped" using the `Collections.synchronizedSortedMap` method. This is best done at creation time, to prevent accidental unsynchronized access to the map:

```
SortedMap m = Collections.synchronizedSortedMap(new TreeMap(...));
```

The iterators returned by the `iterator` method of the collections returned by all of this class's "collection view methods" are *fail-fast*: if the map is structurally modified at any time after the iterator is created, in any way except through the iterator's own `remove` method, the iterator will throw a `ConcurrentModificationException`. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Note that the fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw `ConcurrentModificationException` on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: *the fail-fast behavior of iterators should be used only to detect bugs.*

All `Map.Entry` pairs returned by methods in this class and its views represent snapshots of mappings at the time they were produced. They do **not** support the `Entry.setValue` method. (Note however that it is possible to change mappings in the associated map using `put`.)

This class is a member of the Java Collections Framework.

Since:

1.2

See Also:

`Map`, `HashMap`, `Hashtable`, `Comparable`, `Comparator`, `Collection`, `Serialized Form`

Nested Class Summary

Nested classes/interfaces inherited from class java.util.AbstractMap

`AbstractMap.SimpleEntry<K,V>`, `AbstractMap.SimpleImmutableEntry<K,V>`

Constructor Summary

Constructors

Constructor and Description

`TreeMap()`
Constructs a new, empty tree map, using the natural ordering of its keys.

`TreeMap(Comparator<? super K> comparator)`
Constructs a new, empty tree map, ordered according to the given comparator.

`TreeMap(Map<? extends K,? extends V> m)`
Constructs a new tree map containing the same mappings as the given map, ordered according to the *natural ordering* of its keys.

`TreeMap(SortedMap<K,? extends V> m)`
Constructs a new tree map containing the same mappings and using the same ordering as the specified sorted map.

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
`Map.Entry<K,V>`	`ceilingEntry(K key)` Returns a key-value mapping associated with the least key greater than or equal to the given key, or `null` if there is no such key.
`K`	`ceilingKey(K key)` Returns the least key greater than or equal to the given key, or `null` if there is no such key.
`void`	`clear()` Removes all of the mappings from this map.
`Object`	`clone()` Returns a shallow copy of this `TreeMap` instance.
`Comparator<? super K>`	`comparator()` Returns the comparator used to order the keys in this map, or `null` if this map uses the **natural ordering** of its keys.
`boolean`	`containsKey(Object key)` Returns `true` if this map contains a mapping for the specified key.
`boolean`	`containsValue(Object value)` Returns `true` if this map maps one or more keys to the specified value.
`NavigableSet<K>`	`descendingKeySet()` Returns a reverse order `NavigableSet` view of the keys contained in this map.
`NavigableMap<K,V>`	`descendingMap()` Returns a reverse order view of the mappings contained in this map.
`Set<Map.Entry<K,V>>`	`entrySet()` Returns a `Set` view of the mappings contained in this map.
`Map.Entry<K,V>`	`firstEntry()` Returns a key-value mapping associated with the least key in this map, or `null` if the map is empty.
`K`	`firstKey()` Returns the first (lowest) key currently in this map.
`Map.Entry<K,V>`	`floorEntry(K key)` Returns a key-value mapping associated with the greatest key less than or equal to the given key, or `null` if there is no such key.
`K`	`floorKey(K key)` Returns the greatest key less than or equal to the given key, or `null` if there is no such key.

	or `null` if there is no such key.
`void`	`forEach(BiConsumer<? super K,? super V> action)` Performs the given action for each entry in this map until all entries have been processed or the action throws an exception.
`V`	`get(Object key)` Returns the value to which the specified key is mapped, or `null` if this map contains no mapping for the key.
`SortedMap<K,V>`	`headMap(K toKey)` Returns a view of the portion of this map whose keys are strictly less than `toKey`.
`NavigableMap<K,V>`	`headMap(K toKey, boolean inclusive)` Returns a view of the portion of this map whose keys are less than (or equal to, if `inclusive` is true) `toKey`.
`Map.Entry<K,V>`	`higherEntry(K key)` Returns a key-value mapping associated with the least key strictly greater than the given key, or `null` if there is no such key.
`K`	`higherKey(K key)` Returns the least key strictly greater than the given key, or `null` if there is no such key.
`Set<K>`	`keySet()` Returns a `Set` view of the keys contained in this map.
`Map.Entry<K,V>`	`lastEntry()` Returns a key-value mapping associated with the greatest key in this map, or `null` if the map is empty.
`K`	`lastKey()` Returns the last (highest) key currently in this map.
`Map.Entry<K,V>`	`lowerEntry(K key)` Returns a key-value mapping associated with the greatest key strictly less than the given key, or `null` if there is no such key.
`K`	`lowerKey(K key)` Returns the greatest key strictly less than the given key, or `null` if there is no such key.
`NavigableSet<K>`	`navigableKeySet()` Returns a `NavigableSet` view of the keys contained in this map.
`Map.Entry<K,V>`	`pollFirstEntry()` Removes and returns a key-value mapping associated with the least key in this map, or `null` if the map is empty.
`Map.Entry<K,V>`	`pollLastEntry()` Removes and returns a key-value mapping associated with the greatest key in this map, or `null` if the map is empty.

	the greatest key in this map, or null if the map is empty.
V	**put**(**K** key, **V** value) Associates the specified value with the specified key in this map.
void	**putAll**(**Map**<? extends **K**,? extends **V**> map) Copies all of the mappings from the specified map to this map.
V	**remove**(**Object** key) Removes the mapping for this key from this TreeMap if present.
V	**replace**(**K** key, **V** value) Replaces the entry for the specified key only if it is currently mapped to some value.
boolean	**replace**(**K** key, **V** oldValue, **V** newValue) Replaces the entry for the specified key only if currently mapped to the specified value.
void	**replaceAll**(**BiFunction**<? super **K**,? super **V**,? extends **V**> function) Replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
int	**size**() Returns the number of key-value mappings in this map.
NavigableMap<**K**,**V**>	**subMap**(**K** fromKey, boolean fromInclusive, **K** toKey, boolean toInclusive) Returns a view of the portion of this map whose keys range from fromKey to toKey.
SortedMap<**K**,**V**>	**subMap**(**K** fromKey, **K** toKey) Returns a view of the portion of this map whose keys range from fromKey, inclusive, to toKey, exclusive.
SortedMap<**K**,**V**>	**tailMap**(**K** fromKey) Returns a view of the portion of this map whose keys are greater than or equal to fromKey.
NavigableMap<**K**,**V**>	**tailMap**(**K** fromKey, boolean inclusive) Returns a view of the portion of this map whose keys are greater than (or equal to, if inclusive is true) fromKey.
Collection<**V**>	**values**() Returns a **Collection** view of the values contained in this map.

Methods inherited from class java.util.AbstractMap

equals, hashCode, isEmpty, toString