𝕲𝖗𝖎𝖒𝖔𝖎𝖗𝖊'𝖑 𝕾𝖙𝖆𝖓𝖉𝖆𝖗𝖉 𝕮𝖔𝖉𝖊 𝕷𝖎𝖇𝖗𝖆𝖗𝖞 [*]

Shanghai Jiao Tong University

Dated: 2017 年 10 月 17 日

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

```cpp
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;
```

```
29              for(int j(0); j < m; j++) {
30                  a[i] = (a[i] + (long long)c[j] * a[i + j - m]) % p;
31              }
32          }
33          for(int j(0); j < m; j++) {
34              b[j] = 0;
35              for(int i(0); i < m; i++) {
36                  b[j] = (b[j] + v[i] * a[i + j]) % p;
37              }
38          }
39          for(int j(0); j < m; j++) {
40              a[j] = b[j];
41          }
42      }
```

任意模数快速傅里叶变换

```
1  // double 精度对 10^9 + 7 取模最多可以做到 2^20
2  const int MOD = 1000003;
3  const double PI = acos(-1);
4  typedef complex<double> Complex;
5  const int N = 65536, L = 15, MASK = (1 << L) - 1;
6  Complex w[N];
7  void FFTInit() {
8      for (int i = 0; i < N; ++i)
9          w[i] = Complex(cos(2 * i * PI / N), sin(2 * i * PI / N));
10 }
11 void FFT(Complex p[], int n) {
12     for (int i = 1, j = 0; i < n - 1; ++i) {
13         for (int s = n; j ^= s >>= 1, ~j & s;);
14         if (i < j) swap(p[i], p[j]);
15     }
16     for (int d = 0; (1 << d) < n; ++d) {
17         int m = 1 << d, m2 = m * 2, rm = n >> (d + 1);
18         for (int i = 0; i < n; i += m2) {
19             for (int j = 0; j < m; ++j) {
20                 Complex &p1 = p[i + j + m], &p2 = p[i + j];
21                 Complex t = w[rm * j] * p1;
22                 p1 = p2 - t, p2 = p2 + t;
23             } } }
24 }
25 Complex A[N], B[N], C[N], D[N];
26 void mul(int a[N], int b[N]) {
27     for (int i = 0; i < N; ++i) {
28         A[i] = Complex(a[i] >> L, a[i] & MASK);
```

```
29        B[i] = Complex(b[i] >> L, b[i] & MASK);
30    }
31    FFT(A, N), FFT(B, N);
32    for (int i = 0; i < N; ++i) {
33        int j = (N - i) % N;
34        Complex da = (A[i] - conj(A[j])) * Complex(0, -0.5),
35               db = (A[i] + conj(A[j])) * Complex(0.5, 0),
36               dc = (B[i] - conj(B[j])) * Complex(0, -0.5),
37               dd = (B[i] + conj(B[j])) * Complex(0.5, 0);
38        C[j] = da * dd + da * dc * Complex(0, 1);
39        D[j] = db * dd + db * dc * Complex(0, 1);
40    }
41    FFT(C, N), FFT(D, N);
42    for (int i = 0; i < N; ++i) {
43        long long da = (long long)(C[i].imag() / N + 0.5) % MOD,
44                 db = (long long)(C[i].real() / N + 0.5) % MOD,
45                 dc = (long long)(D[i].imag() / N + 0.5) % MOD,
46                 dd = (long long)(D[i].real() / N + 0.5) % MOD;
47        a[i] = ((dd << (L * 2)) + ((db + dc) << L) + da) % MOD;
48    }
49 }
```

快速傅里叶变换

```
1  int prepare(int n) {
2      int len = 1;
3      for (; len <= 2 * n; len <<= 1);
4      for (int i = 0; i < len; i++) {
5          e[0][i] = Complex(cos(2 * pi * i / len), sin(2 * pi * i / len));
6          e[1][i] = Complex(cos(2 * pi * i / len), -sin(2 * pi * i / len));
7      }
8      return len;
9  }
10 void DFT(Complex *a, int n, int f) {
11     for (int i = 0, j = 0; i < n; i++) {
12         if (i > j) std::swap(a[i], a[j]);
13         for (int t = n >> 1; (j ^= t) < t; t >>= 1);
14     }
15     for (int i = 2; i <= n; i <<= 1)
16         for (int j = 0; j < n; j += i)
17             for (int k = 0; k < (i >> 1); k++) {
18                 Complex A = a[j + k];
19                 Complex B = e[f][n / i * k] * a[j + k + (i >> 1)];
20                 a[j + k] = A + B;
21                 a[j + k + (i >> 1)] = A - B;
```

```
22            }
23        if (f == 1) {
24            for (int i = 0; i < n; i++)
25                a[i].a /= n;
26        }
27 }
```

闪电数论变换与魔力 CRT

```
1  #define meminit(A, l, r) memset(A + (l), 0, sizeof(*A) * ((r) - (l)))
2  #define memcopy(B, A, l, r) memcpy(B, A + (l), sizeof(*A) * ((r) - (l)))
3  void DFT(int *a, int n, int f) { //f=1 逆 DFT
4      for (register int i = 0, j = 0; i < n; i++) {
5          if (i > j) std::swap(a[i], a[j]);
6          for (register int t = n >> 1; (j ^= t) < t; t >>= 1);
7      }
8      for (register int i = 2; i <= n; i <<= 1) {
9          static int exp[MAXN];
10         exp[0] = 1; exp[1] = fpm(PRT, (MOD - 1) / i, MOD);
11         if (f == 1) exp[1] = fpm(exp[1], MOD - 2, MOD);
12         for (register int k = 2; k < (i >> 1); k++) {
13             exp[k] = 1ll * exp[k - 1] * exp[1] % MOD;
14         }
15         for (register int j = 0; j < n; j += i) {
16             for (register int k = 0; k < (i >> 1); k++) {
17                 register int &pA = a[j + k], &pB = a[j + k + (i >> 1)];
18                 register long long B = 1ll * pB * exp[k];
19                 pB = (pA - B) % MOD;
20                 pA = (pA + B) % MOD;
21             }
22         }
23     }
24     if (f == 1) {
25         register int rev = fpm(n, MOD - 2, MOD);
26         for (register int i = 0; i < n; i++) {
27             a[i] = 1ll * a[i] * rev % MOD;
28             if (a[i] < 0) { a[i] += MOD; }
29         }
30     }
31 }
32 // 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
33 // 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
34 int CRT(int *a) {
35     static int x[3];
36     for (int i = 0; i < 3; i++) {
```

```
37          x[i] = a[i];
38          for (int j = 0; j < i; j++) {
39              int t = (x[i] - x[j] + FFT[i] -> MOD) % FFT[i] -> MOD;
40              if (t < 0) t += FFT[i] -> MOD;
41              x[i] = 1LL * t * inv[j][i] % FFT[i] -> MOD;
42          }
43      }
44      int sum = 1, ret = x[0] % MOD;
45      for (int i = 1; i < 3; i ++) {
46          sum = 1LL * sum * FFT[i - 1] -> MOD % MOD;
47          ret += 1LL * x[i] * sum % MOD;
48          if(ret >= MOD) ret -= MOD;
49      }
50      return ret;
51  }
52  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
53      for (int j = 0; j < 3; j++)
54          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 $a * b \equiv 1(\mod x^n)$

```
1  void getInv(int *a, int *b, int n) {
2      static int tmp[MAXN];
3      b[0] = fpm(a[0], MOD - 2, MOD);
4      for (int c = 2, M = 1; c < (n << 1); c <<= 1) {
5          for (; M <= 3 * (c - 1); M <<= 1);
6          meminit(b, c, M);
7          meminit(tmp, c, M);
8          memcopy(tmp, a, 0, c);
9          DFT(tmp, M, 0);
10         DFT(b, M, 0);
11         for (int i = 0; i < M; i++) {
12             b[i] = 1ll * b[i] * (2ll - 1ll * tmp[i] * b[i] % MOD + MOD) % MOD;
13         }
14         DFT(b, M, 1);
15         meminit(b, c, M);
16     }
17  }
```

多项式除法

d is quotient and r is remainder

```cpp
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 $b \equiv e^a(\mod x^n)$ or $b \equiv \ln a(\mod x^n)$

```cpp
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;
    }
```

```
 5        b[n - 1] = 0;
 6    }
 7    void getInt(int *a, int *b, int n) { // 多项式取积分，积分常数为 0
 8        static int inv[MAXN];
 9        inv[1] = 1;
10        for (int i = 2; i < n; i++) {
11            inv[i] = 1ll * (MOD - MOD / i) * inv[MOD % i] % MOD;
12        }
13        b[0] = 0;
14        for (int i = 1; i < n; i++) {
15            b[i] = 1ll * a[i - 1] * inv[i] % MOD;
16        }
17    }
18    void getLn(int *a, int *b, int n) {
19        static int inv[MAXN], d[MAXN];
20        int M = 1;
21        for (; M <= 2 * (n - 1); M <<= 1);
22        getInv(a, inv, n);
23        getDiff(a, d, n);
24        meminit(d, n, M);
25        meminit(inv, n, M);
26        DFT(d, M, 0); DFT(inv, M, 0);
27        for (int i = 0; i < M; i++) {
28            d[i] = 1ll * d[i] * inv[i] % MOD;
29        }
30        DFT(d, M, 1);
31        getInt(d, b, n);
32    }
33    void getExp(int *a, int *b, int n) {
34        static int ln[MAXN], tmp[MAXN];
35        b[0] = 1;
36        for (int c = 2, M = 1; c < (n << 1); c <<= 1) {
37            for (; M <= 2 * (c - 1); M <<= 1);
38            int bound = std::min(c, n);
39            memcopy(tmp, a, 0, bound);
40            meminit(tmp, bound, M);
41            meminit(b, c, M);
42            getLn(b, ln, c);
43            meminit(ln, c, M);
44            DFT(b, M, 0);
45            DFT(tmp, M, 0);
46            DFT(ln, M, 0);
47            for (int i = 0; i < M; i++) {
48                b[i] = 1ll * b[i] * (1ll - ln[i] + tmp[i] + MOD) % MOD;
49            }
50            DFT(b, M, 1);
```

```
51        meminit(b, c, M);
52    }
53 }
```

快速沃尔什变换

```
1  void FWT(LL a[],int n,int ty){
2      for(int d=1;d<n;d<<=1){
3          for(int m=(d<<1),i=0;i<n;i+=m){
4              if(ty==1){
5                  for(int j=0;j<d;j++){
6                      LL x=a[i+j],y=a[i+j+d];
7                      a[i+j]=x+y;
8                      a[i+j+d]=x-y;
9                      //xor:a[i+j]=x+y,a[i+j+d]=x-y;
10                     //and:a[i+j]=x+y;
11                     //or:a[i+j+d]=x+y;
12                 }
13             }else{
14                 for(int j=0;j<d;j++){
15                     LL x=a[i+j],y=a[i+j+d];
16                     a[i+j]=(x+y)/2;
17                     a[i+j+d]=(x-y)/2;
18                     //xor:a[i+j]=(x+y)/2,a[i+j+d]=(x-y)/2;
19                     //and:a[i+j]=x-y;
20                     //or:a[i+j+d]=y-x;
21                 }
22             }
23         }
24     }
25 }
26     FWT(a,1<<n,1);
27     FWT(b,1<<n,1);
28     for(int i=0;i<(1<<n);i++)
29         c[i]=a[i]*b[i];
30     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);
```

```
10          if(res==n-1) return 1;
11      }
12      return 0; // n is not a strong pseudo prime
13 }
14 bool isprime(const long long &n) {
15      if(n==2)
16          return true;
17      if(n<2 || n%2==0)
18          return false;
19      for(int i=0;i<12&&BASE[i]<n;i++){
20          if(!check(n,BASE[i]))
21              return false;
22      }
23      return true;
24 }
```

Pollard-rho.cpp

```
1  LL prho(LL n,LL c){
2      LL i=1,k=2,x=rand()%(n-1)+1,y=x;
3      while(1){
4          i++;x=(x*x%n+c)%n;
5          LL d=__gcd((y-x+n)%n,n);
6          if(d>1&&d<n)return d;
7          if(y==x)return n;
8          if(i==k)y=x,k<<=1;
9      }
10 }
11 void factor(LL n,vector<LL>&fat){
12      if(n==1)return;
13      if(isprime(n)){
14          fat.push_back(n);
15          return;
16      }LL p=n;
17      while(p>=n)p=prho(p,rand()%(n-1)+1);
18      factor(p,fat);
19      factor(n/p,fat);
20 }
```

非互质 CRT

first is remainder, second is module

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

```cpp
}
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

```cpp
//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

```
1  // x_{k+1} = x_0 x_k + n y_0 y_k
2  // y_{k+1} = x_0 y_k + y_0 x_k
3  // n is not the index of which you want
4  pair<ll, ll> pell(ll n) {
5      static ll p[N], q[N], g[N], h[N], a[N];
6      p[1] = q[0] = h[1] = 1; p[0] = q[1] = g[1] = 0;
7      a[2] = (ll)(floor(sqrtl(n) + 1e-7L));
8      for(int i = 2; ; i ++) {
9          g[i] = -g[i - 1] + a[i] * h[i - 1];
10         h[i] = (n - g[i] * g[i]) / h[i - 1];
11         a[i + 1] = (g[i] + a[2]) / h[i];
12         p[i] = a[i] * p[i - 1] + p[i - 2];
13         q[i] = a[i] * q[i - 1] + q[i - 2];
14         if(p[i] * p[i] - n * q[i] * q[i] == 1)
15             return {p[i], q[i]};
16     }
17 } // x^2 - n * y^2 = 1 最小正整数根, n 为完全平方数时无解
```

Simpson

```
1  // 三次函数，两倍精度拟合
2  // error = (r-l)^5 / 6480 |f^{(4)}|
3  // ∫_a^b f(x) dx ≈ (b-a)/8 [f(a) + 3f((2a+b)/3) + 3f((a+2b)/3) + f(b)]
4  // 三次函数拟合  error = 1/90 (r-l)^5 / 2 |f^{(4)}|
5  d simpson(d fl,d fr,d fmid,d l,d r) {
6      return (fl+fr+4.0*fmid)*(r-l)/6.0; }
7  d rsimpson(d slr,d fl,d fr,d fmid,d l,d r) {
8      d mid = (l+r)/2,fml = f((l+mid)/2),fmr = f((mid+r)/2);
9      d slm = simpson(fl,fmid,fml,l,mid);
10     d smr = simpson(fmid,fr,fmr,mid,r);
11     if(fabs(slr - smr - slm) / slr < eps)return slm + smr;
12     return rsimpson(slm,fl,fmid,fml,l,mid)+
13         rsimpson(smr,fmid,fr,fmr,mid,r);
14 }
```

解一元三次方程

听说极端情况精度不够

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

```
4  double q(2. * k * k * k / 27 - k * m / 3. + n);
5  Complex omega[3] = {Complex(1, 0), Complex(-0.5, 0.5 * sqrt(3)), Complex(-0.5, -0.5 *
   ↪ sqrt(3))};
6  Complex r1, r2;
7  double delta(q * q / 4 + p * p * p / 27);
8  if (delta > 0) {
9      r1 = cubrt(-q / 2. + sqrt(delta));
10     r2 = cubrt(-q / 2. - sqrt(delta));
11 } else {
12     r1 = pow(-q / 2. + pow(Complex(delta), 0.5), 1. / 3);
13     r2 = pow(-q / 2. - pow(Complex(delta), 0.5), 1. / 3);
14 }
15 for(int _(0); _ < 3; _++) {
16     Complex x = -k / 3. + r1 * omega[_ * 1] + r2 * omega[_ * 2 % 3];
17 }
```

线段下整点

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

```
1  LL solve(LL n,LL a,LL b,LL m){
2      if(b==0) return n*(a/m);
3      if(a>=m) return n*(a/m)+solve(n,a%m,b,m);
4      if(b>=m) return (n-1)*n/2*(b/m)+solve(n,a,b%m,m);
5      return solve((a+b*n)/m,(a+b*n)%m,m,b);
6  }
```

线性同余不等式

```
1  // Find the minimal non-negtive solutions for l ≤ d · x mod m ≤ r
2  // 0 ≤ d,l,r < m;l ≤ r,O(log n)
3  ll cal(ll m, ll d, ll l, ll r) {
4      if (l == 0) return 0;
5      if (d == 0) return MXL; // 无解
6      if (d * 2 > m) return cal(m, m - d, m - r, m - l);
7      if ((l - 1) / d < r / d) return (l - 1) / d + 1;
8      ll k = cal(d, (-m % d + d) % d, l % d, r % d);
9      return k == MXL ? MXL : (k * m + l - 1) / d + 1; // 无解 2
10 }
```

EX-BSGS -zzq

```
1  /*
2   * EX_BSGS
```

```
3   * a^x = b (mod p)
4   * p may not be a prime
5   */
6
7  ll qpow(ll a, ll x, ll Mod) {
8      ll res = 1;
9      for (; x; x >>= 1) {
10         if (x & 1) res = res * a % Mod;
11         a = a * a % Mod;
12     }
13     return res;
14 }
15
16 std::unordered_map<int, int> mp;
17
18 ll exbsgs(ll a, ll b, ll p) {
19     if (b == 1) return 0;
20     ll t, d = 1, k = 0;
21     while ((t = std::__gcd(a, p)) != 1) {
22         if (b % t) return -1;
23         ++k, b /= t, p /= t, d = d * (a / t) % p;
24         if (b == d) return k;
25     }
26     mp.clear();
27     ll m = std::ceil(std::sqrt(p));
28     ll a_m = qpow(a, m, p);
29     ll mul = b;
30     for (ll j = 1; j <= m; ++j) {
31         mul = mul * a % p;
32         mp[mul] = j;
33     }
34     for (ll i = 1; i <= m; ++i) {
35         d = d * a_m % p;
36         if (mp.count(d)) return i * m - mp[d] + k;
37     }
38     return -1;
39 }
```

EX-BSGS -zky

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

```
 6      for(int i=0;i<=m;i++){
 7          if(hash.count(b))return i*m+hash[b];
 8          b=b*v%p;
 9      }return -1;
10 }
11
12 LL solve2(LL a,LL b,LL p){
13     //a^x=b (mod p)
14     b%=p;
15     LL e=1%p;
16     for(int i=0;i<100;i++){
17         if(e==b)return i;
18         e=e*a%p;
19     }
20     int r=0;
21     while(gcd(a,p)!=1){
22         LL d=gcd(a,p);
23         if(b%d)return -1;
24         p/=d;b/=d;b=b*inv(a/d,p);
25         r++;
26     }LL res=BSGS(a,b,p);
27     if(res==-1)return -1;
28     return res+r;
29 }
```

分治乘法

```
1 (a+b)(c+d) = ac+(bc+ad)+bd = 2ac-(a-b)(c-d)+2bd
2
3 x = x^m m=(n+1)/2
4 (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

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

```
11
12          ans=ans*i%P;
13      }
14      return make_pair(cnt,ans);
15  }
16  pair<LL,LL> calc(LL n,LL p,LL k){
17      prod=1;P=pow(p,k,1e18);
18      for(int i=1;i<P;i++)if(i%p)prod=prod*i%P;
19      pair<LL,LL> res=comput(n,p,k);
20  //  res.second=res.second*pow(p,res.first%k,P)%P;
21  //  res.first-=res.first%k;
22      return res;
23  }
24  LL calc(LL n,LL m,LL p,LL k){
25      pair<LL,LL>A,B,C;
26      LL P=pow(p,k,1e18);
27      A=calc(n,p,k);
28      B=calc(m,p,k);
29      C=calc(n-m,p,k);
30      LL ans=1;
31      ans=pow(p,A.first-B.first-C.first,P);
32      ans=ans*A.second%P*inv(B.second,P)%P*inv(C.second,P)%P;
33      return ans;
34  }
```

线性筛

```
1  void sieve(){
2      f[1]=mu[1]=phi[1]=1;
3      for(int i=2;i<maxn;i++){
4          if(!minp[i]){
5              minp[i]=i;
6              minpw[i]=i;
7              mu[i]=-1;
8              phi[i]=i-1;
9              f[i]=i-1;
10             p[++p[0]]=i;//Case 1 prime
11         }
12         for(int j=1;j<=p[0]&&(LL)i*p[j]<maxn;j++){
13             minp[i*p[j]]=p[j];
14             if(i%p[j]==0){
15                 //Case 2 not coprime
16                 minpw[i*p[j]]=minpw[i]*p[j];
17                 phi[i*p[j]]=phi[i]*p[j];
18                 mu[i*p[j]]=0;
```

```
19              if(i==minpw[i]){
20                  f[i*p[j]]=i*p[j]-i;//Special Case for f(p^k)
21              }else{
22                  f[i*p[j]]=f[i/minpw[i]]*f[minpw[i]*p[j]];
23              }
24              break;
25          }else{
26              //Case 3 coprime
27              minpw[i*p[j]]=p[j];
28              f[i*p[j]]=f[i]*f[p[j]];
29              phi[i*p[j]]=phi[i]*(p[j]-1);
30              mu[i*p[j]]=-mu[i];
31          }
32      }
33  }
34 }
```

Chapter 3

图论

图论基础

```cpp
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 -zzq

```cpp
typedef std::pair<int, int> pii;
#define mkpair std::make_pair

int n, m;
std::vector<int> G[MAXN];
```

```
6
7   int dfn[MAXN], low[MAXN], bcc_id[MAXN], bcc_cnt, stamp;
8   bool iscut[MAXN];
9
10  std::vector<int> bcc[MAXN]; // Unnecessary
11
12  pii stk[MAXN]; int stk_top;
13  // Use a handwritten structure to get higher efficiency
14
15  void Tarjan(int now, int fa) {
16      int child = 0;
17      dfn[now] = low[now] = ++stamp;
18      for (int to: G[now]) {
19          if (!dfn[to]) {
20              stk[++stk_top] = mkpair(now, to); ++child;
21              Tarjan(to, now);
22              low[now] = std::min(low[now], low[to]);
23              if (low[to] >= dfn[now]) {
24                  iscut[now] = 1;
25                  bcc[++bcc_cnt].clear();
26                  while (1) {
27                      pii tmp = stk[stk_top--];
28                      if (bcc_id[tmp.first] != bcc_cnt) {
29                          bcc[bcc_cnt].push_back(tmp.first);
30                          bcc_id[tmp.first] = bcc_cnt;
31                      }
32                      if (bcc_id[tmp.second] != bcc_cnt) {
33                          bcc[bcc_cnt].push_back(tmp.second);
34                          bcc_id[tmp.second] = bcc_cnt;
35                      }
36                      if (tmp.first == now && tmp.second == to)
37                          break;
38                  }
39              }
40          }
41          else if (dfn[to] < dfn[now] && to != fa) {
42              stk[++stk_top] = mkpair(now, to);
43              low[now] = std::min(low[now], dfn[to]);
44          }
45      }
46      if (!fa && child == 1)
47          iscut[now] = 0;
48  }
49
50  void PBCC() {
51      memset(dfn, 0, sizeof dfn);
```

```
52        memset(low, 0, sizeof low);
53        memset(iscut, 0, sizeof iscut);
54        memset(bcc_id, 0, sizeof bcc_id);
55        stamp = bcc_cnt = stk_top = 0;
56
57        for (int i = 1; i <= n; ++i)
58            if (!dfn[i]) Tarjan(i, 0);
59 }
```

坚固无敌的边双 **-zzq**

```
1  int n, m;
2  int head[MAXN], nxt[MAXM << 1], to[MAXM << 1], ed;
3  // Opposite edge exists, set head[] to -1.
4
5  int dfn[MAXN], low[MAXN], bcc_id[MAXN], bcc_cnt, stamp;
6  bool isbridge[MAXM << 1], vis[MAXN];
7
8  std::vector<int> bcc[MAXN];
9
10 void Tarjan(int now, int fa) {
11     dfn[now] = low[now] = ++stamp;
12     for (int i = head[now]; ~i; i = nxt[i]) {
13         if (!dfn[to[i]]) {
14             Tarjan(to[i], now);
15             low[now] = std::min(low[now], low[to[i]]);
16             if (low[to[i]] > dfn[now])
17                 isbridge[i] = isbridge[i ^ 1] = 1;
18         }
19         else if (dfn[to[i]] < dfn[now] && to[i] != fa)
20             low[now] = std::min(low[now], dfn[to[i]]);
21     }
22 }
23
24 void DFS(int now) {
25     vis[now] = 1;
26     bcc_id[now] = bcc_cnt;
27     bcc[bcc_cnt].push_back(now);
28     for (int i = head[now]; ~i; i = nxt[i]) {
29         if (isbridge[i]) continue;
30         if (!vis[to[i]]) DFS(to[i]);
31     }
32 }
33
34 void EBCC() {
```

```
35    memset(dfn, 0, sizeof dfn);
36    memset(low, 0, sizeof low);
37    memset(isbridge, 0, sizeof isbridge);
38    memset(bcc_id, 0, sizeof bcc_id);
39    bcc_cnt = stamp = 0;
40
41    for (int i = 1; i <= n; ++i)
42        if (!dfn[i]) Tarjan(i, 0);
43
44    memset(vis, 0, sizeof vis);
45    for (int i = 1; i <= n; ++i)
46        if (!vis[i]) {
47            ++bcc_cnt;
48            DFS(i);
49        }
50 }
```

坚固无敌的点双 -jzh

```
1  const bool BCC_VERTEX = 0, BCC_EDGE = 1;
2  struct BCC {  // N = N0 + M0. Remember to call init(&raw_graph).
3      Graph *g, forest; // g is raw graph ptr.
4      int dfn[N], DFN, low[N];
5      int stack[N], top;
6      int expand_to[M];      // Where edge i is expanded to in expaned graph.
7      // Vertex i expaned to i.
8      int compress_to[N];  // Where vertex i is compressed to.
9      bool cut[N], compress_cut[N], branch[M], vis[N], flag;
10     //std::vector<int> BCC_component[N];  // Cut vertex belongs to none.
11     __inline void init(Graph *raw_graph) {
12         g = raw_graph;
13     }
14     void DFS(int u, int pe) {
15         dfn[u] = low[u] = ++DFN; cut[u] = false;
16         if (!~g->adj[u]) {
17             cut[u] = 1;
18             compress_to[u] = forest.new_node();
19             compress_cut[compress_to[u]] = 1;
20         }
21         for (int e = g->adj[u]; ~e; e = g->nxt[e]) {
22             int v = g->v[e];
23             if ((e ^ pe) > 1 && dfn[v] > 0 && dfn[v] < dfn[u]) {
24                 stack[top++] = e;
25                 low[u] = std::min(low[u], dfn[v]);
26             }
```

```
27              else if (!dfn[v]) {
28                  stack[top++] = e; branch[e] = 1;
29                  DFS(v, e);
30                  low[u] = std::min(low[v], low[u]);
31                  if (low[v] >= dfn[u]) {
32                      if ((pe == -1 && flag || pe != -1) && !cut[u]) {
33                          cut[u] = 1;
34                          compress_to[u] = forest.new_node();
35                          compress_cut[compress_to[u]] = 1;
36                      }
37                      int cc = forest.new_node();
38                      if (cut[u]) {
39                          forest.bi_ins(compress_to[u], cc);
40                      }
41                      compress_cut[cc] = 0;
42                      //BCC_component[cc].clear();
43                      do {
44                          int cur_e = stack[--top];
45                          compress_to[expand_to[cur_e]] = cc;
46                          compress_to[expand_to[cur_e^1]] = cc;
47                          if (branch[cur_e]) {
48                              int v = g->v[cur_e];
49                              if (cut[v]) {
50                                  forest.bi_ins(cc, compress_to[v]);
51                              } else {
52                                  //BCC_component[cc].push_back(v);
53                                  compress_to[v] = cc;
54                              }
55                          }
56                      } while (stack[top] != e);
57                      if (pe == -1 && !flag) {
58                          compress_to[u] = cc;
59                      }
60                  }
61              }
62          }
63      }
64      inline bool dfs(int u, int pe) {
65          vis[u] = 1;
66          int d = 0;
67          for (int e = g->adj[u]; ~e; e = g->nxt[e]) {
68              int v = g -> v[e];
69              if (!vis[v]) {
70                  ++d;
71                  dfs(v, e);
72              }
```

```
73          }
74          return pe == -1 ? d > 1 : 0;
75      }
76      void solve() {
77          forest.init(g->base);
78          int n = g->n;
79          for (int i = 0; i < g->e; i++) {
80              expand_to[i] = g->new_node();
81          }
82          memset(vis + g -> base, 0, sizeof(*vis) * n);
83          memset(branch, 0, sizeof(*branch) * g->e);
84          memset(dfn + g->base, 0, sizeof(*dfn) * n); DFN = 0;
85          for (int i = 0; i < n; i++)
86              if (!dfn[i + g->base]) {
87                  top = 0;
88                  flag = dfs(i + g -> base, -1);
89                  DFS(i + g->base, -1);
90              }
91      }
92 } bcc;
```

坚固无敌的边双 -jzh

```
1  struct BCC {
2      Graph *g, forest;
3      int dfn[N], low[N], stack[N], tot[N], belong[N], vis[N], top, dfs_clock;
4      // tot[] is the size of each BCC, belong[] is the BCC that each node belongs to
5      pair<int, int > ori[M]; // bridge in raw_graph(raw node)
6      bool is_bridge[M];
7      __inline void init(Graph *raw_graph) {
8          g = raw_graph;
9          memset(is_bridge, false, sizeof(*is_bridge) * g -> e);
10         memset(vis + g -> base, 0, sizeof(*vis) * g -> n);
11     }
12     void tarjan(int u, int from) {
13         dfn[u] = low[u] = ++dfs_clock; vis[u] = 1; stack[++top] = u;
14         for (int p = g -> adj[u]; ~p; p = g -> nxt[p]) {
15             if ((p ^ 1) == from) continue;
16             int v = g -> v[p];
17             if (vis[v]) {
18                 if (vis[v] == 1) low[u] = min(low[u], dfn[v]);
19             } else {
20                 tarjan(v, p);
21                 low[u] = min(low[u], low[v]);
22                 if (low[v] > dfn[u]) is_bridge[p / 2] = true;
```

```
23                }
24            }
25            if (dfn[u] != low[u]) return;
26            tot[forest.new_node()] = 0;
27            do {
28                belong[stack[top]] = forest.n;
29                vis[stack[top]] = 2;
30                tot[forest.n]++;
31                --top;
32            } while (stack[top + 1] != u);
33        }
34    void solve() {
35        forest.init(g -> base);
36        int n = g -> n;
37        for (int i = 0; i < n; ++i)
38            if (!vis[i + g -> base]) {
39                top = dfs_clock = 0;
40                tarjan(i + g -> base, -1);
41            }
42        for (int i = 0; i < g -> e / 2; ++i)
43            if (is_bridge[i]) {
44                int e = forest.e;
45                forest.bi_ins(belong[g -> v[i * 2]], belong[g -> v[i * 2 + 1]], g -> w[i *
    ↪ 2]);
46                ori[e] = make_pair(g -> v[i * 2 + 1], g -> v[i * 2]);
47                ori[e + 1] = make_pair(g -> v[i * 2], g -> v[i * 2 + 1]);
48            }
49    }
50 } bcc;
```

闪电二分图匹配

```
1 int matchx[N], matchy[N], level[N];
2 vector<int> edge[N];
3 bool dfs(int x) {
4     for (int i = 0; i < (int)edge[x].size(); ++i) {
5         int y = edge[x][i];
6         int w = matchy[y];
7         if (w == -1 || level[x] + 1 == level[w] && dfs(w)) {
8             matchx[x] = y;
9             matchy[y] = x;
10            return true;
11        }
12    }
13    level[x] = -1;
```

```cpp
14        return false;
15 }
16 int solve() {
17     memset(matchx, -1, sizeof(*matchx) * n);
18     memset(matchy, -1, sizeof(*matchy) * m);
19     for (int ans = 0; ; ) {
20         std::vector<int> q;
21         for (int i = 0; i < n; ++i) {
22             if (matchx[i] == -1) {
23                 level[i] = 0;
24                 q.push_back(i);
25             } else {
26                 level[i] = -1;
27             }
28         }
29         for (int head = 0; head < (int)q.size(); ++head) {
30             int x = q[head];
31             for (int i = 0; i < (int)edge[x].size(); ++i) {
32                 int y = edge[x][i];
33                 int w = matchy[y];
34                 if (w != -1 && level[w] < 0) {
35                     level[w] = level[x] + 1;
36                     q.push_back(w);
37                 }
38             }
39         }
40         int delta = 0;
41         for (int i = 0; i < n; ++i) {
42             if (matchx[i] == -1 && dfs(i)) {
43                 delta++;
44             }
45         }
46         if (delta == 0) {
47             return ans;
48         } else {
49             ans += delta;
50         }
51     }
52 }
```

一般图匹配

```cpp
1 // 0-base, match[u] is linked to u
2 vector<int> lnk[MAXN];
3 int match[MAXN], Queue[MAXN], pred[MAXN], base[MAXN], head, tail, sta, fin, nbase;
```

```
 4  bool inQ[MAXN], inB[MAXN];
 5  inline void push(int u) {
 6      Queue[tail++] = u; inQ[u] = 1;
 7  }
 8  inline int pop() {
 9      return Queue[head++];
10  }
11  inline int FindCA(int u, int v) {
12      static bool inP[MAXN];
13      fill(inP, inP + n, false);
14      while (1) {
15          u = base[u]; inP[u] = 1;
16          if(u == sta) break;
17          u = pred[match[u]];
18      }
19      while (1) {
20          v = base[v];
21          if (inP[v]) break;
22          v = pred[match[v]];
23      }
24      return v;
25  }
26  inline void RT(int u) {
27      int v;
28      while (base[u] != nbase) {
29          v = match[u];
30          inB[base[u]] = inB[base[v]] = 1;
31          u = pred[v];
32          if (base[u] != nbase) pred[u] = v;
33      }
34  }
35  inline void BC(int u, int v) {
36      nbase = FindCA(u, v);
37      fill(inB, inB + n, 0);
38      RT(u); RT(v);
39      if (base[u] != nbase) pred[u] = v;
40      if (base[v] != nbase) pred[v] = u;
41      for (int i = 0; i < n; ++i)
42          if (inB[base[i]]) {
43              base[i] = nbase;
44              if (!inQ[i]) push(i);
45          }
46  }
47  bool FindAP(int u) {
48      bool found = false;
49      for (int i = 0; i < n; ++i) {
```

```
50          pred[i] = -1; base[i] = i; inQ[i] = 0;
51      }
52      sta = u; fin = -1; head = tail = 0; push(sta);
53      while (head < tail) {
54          int u = pop();
55          for (int i = (int)lnk[u].size() - 1; i >= 0; --i) {
56              int v = lnk[u][i];
57              if (base[u] != base[v] && match[u] != v) {
58                  if (v == sta || match[v] >= 0 && pred[match[v]] >= 0) BC(u, v);
59                  else if (pred[v] == -1) {
60                      pred[v] = u;
61                      if (match[v] >= 0) push(match[v]);
62                      else {
63                          fin = v;
64                          return true;
65                      }
66                  }
67              }
68          }
69      }
70      return found;
71  }
72  inline void AP() {
73      int u = fin, v, w;
74      while (u >= 0) {
75          v = pred[u]; w = match[v];
76          match[v] = u; match[u] = v;
77          u = w;
78      }
79  }
80  inline int FindMax() {
81      for (int i = 0; i < n; ++i) match[i] = -1;
82      for (int i = 0; i < n; ++i)
83          if (match[i] == -1 && FindAP(i)) AP();
84      int ans = 0;
85      for (int i = 0; i < n; ++i) {
86          ans += (match[i] != -1);
87      }
88      return ans;
89  }
```

一般最大权匹配

```
1  //maximum weight blossom,  change g[u][v].w to INF - g[u][v].w when minimum weight blossom
      ↪ is needed
2  //type of ans is long long
3  //replace all int to long long if weight of edge is long long
4
5  struct WeightGraph {
6      static const int INF = INT_MAX;
7      static const int MAXN = 400;
8      struct edge{
9          int u, v, w;
10         edge() {}
11         edge(int u, int v, int w): u(u), v(v), w(w) {}
12     };
13     int n, n_x;
14     edge g[MAXN * 2 + 1][MAXN * 2 + 1];
15     int lab[MAXN * 2 + 1];
16     int match[MAXN * 2 + 1], slack[MAXN * 2 + 1], st[MAXN * 2 + 1], pa[MAXN * 2 + 1];
17     int flower_from[MAXN * 2 + 1][MAXN+1], S[MAXN * 2 + 1], vis[MAXN * 2 + 1];
18     vector<int> flower[MAXN * 2 + 1];
19     queue<int> q;
20     inline int e_delta(const edge &e){ // does not work inside blossoms
21         return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
22     }
23     inline void update_slack(int u, int x){
24         if(!slack[x] || e_delta(g[u][x]) < e_delta(g[slack[x]][x]))
25             slack[x] = u;
26     }
27     inline void set_slack(int x){
28         slack[x] = 0;
29         for(int u = 1;u <= n; ++u)
30             if(g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
31                 update_slack(u, x);
32     }
33     void q_push(int x){
34         if(x <= n)q.push(x);
35         else for(size_t i = 0;i < flower[x].size(); i++)
36             q_push(flower[x][i]);
37     }
38     inline void set_st(int x, int b){
39         st[x]=b;
40         if(x > n) for(size_t i = 0;i < flower[x].size(); ++i)
41                 set_st(flower[x][i], b);
42     }
43     inline int get_pr(int b, int xr){
```

```
44        int pr = find(flower[b].begin(), flower[b].end(), xr) - flower[b].begin();
45        if(pr % 2 == 1){
46            reverse(flower[b].begin() + 1, flower[b].end());
47            return (int)flower[b].size() - pr;
48        } else return pr;
49    }
50    inline void set_match(int u, int v){
51        match[u]=g[u][v].v;
52        if(u > n){
53            edge e=g[u][v];
54            int xr = flower_from[u][e.u], pr=get_pr(u, xr);
55            for(int i = 0;i < pr; ++i)
56                set_match(flower[u][i], flower[u][i ^ 1]);
57            set_match(xr, v);
58            rotate(flower[u].begin(), flower[u].begin()+pr, flower[u].end());
59        }
60    }
61    inline void augment(int u, int v){
62        for(; ; ){
63            int xnv=st[match[u]];
64            set_match(u, v);
65            if(!xnv)return;
66            set_match(xnv, st[pa[xnv]]);
67            u=st[pa[xnv]], v=xnv;
68        }
69    }
70    inline int get_lca(int u, int v){
71        static int t=0;
72        for(++t; u || v; swap(u, v)){
73            if(u == 0)continue;
74            if(vis[u] == t)return u;
75            vis[u] = t;
76            u = st[match[u]];
77            if(u) u = st[pa[u]];
78        }
79        return 0;
80    }
81    inline void add_blossom(int u, int lca, int v){
82        int b = n + 1;
83        while(b <= n_x && st[b]) ++b;
84        if(b > n_x) ++n_x;
85        lab[b] = 0, S[b] = 0;
86        match[b] = match[lca];
87        flower[b].clear();
88        flower[b].push_back(lca);
89        for(int x = u, y; x != lca; x = st[pa[y]]) {
```

```
 90              flower[b].push_back(x),
 91              flower[b].push_back(y = st[match[x]]),
 92              q_push(y);
 93          }
 94          reverse(flower[b].begin() + 1, flower[b].end());
 95          for(int x = v, y; x != lca; x = st[pa[y]]) {
 96              flower[b].push_back(x),
 97              flower[b].push_back(y = st[match[x]]),
 98              q_push(y);
 99          }
100          set_st(b, b);
101          for(int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w = 0;
102          for(int x = 1; x <= n; ++x) flower_from[b][x] = 0;
103          for(size_t i = 0 ; i < flower[b].size(); ++i){
104              int xs = flower[b][i];
105              for(int x = 1; x <= n_x; ++x)
106                  if(g[b][x].w == 0 || e_delta(g[xs][x]) < e_delta(g[b][x]))
107                      g[b][x] = g[xs][x], g[x][b] = g[x][xs];
108              for(int x = 1;x <= n; ++x)
109                  if(flower_from[xs][x]) flower_from[b][x] = xs;
110          }
111          set_slack(b);
112      }
113      inline void expand_blossom(int b){ // S[b]  ==  1
114          for(size_t i = 0; i < flower[b].size(); ++i)
115              set_st(flower[b][i], flower[b][i]);
116          int xr = flower_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
117          for(int i = 0; i < pr; i += 2){
118              int xs = flower[b][i], xns = flower[b][i + 1];
119              pa[xs] = g[xns][xs].u;
120              S[xs] = 1, S[xns] = 0;
121              slack[xs] = 0, set_slack(xns);
122              q_push(xns);
123          }
124          S[xr] = 1, pa[xr] = pa[b];
125          for(size_t i = pr + 1;i < flower[b].size(); ++i){
126              int xs = flower[b][i];
127              S[xs] = -1, set_slack(xs);
128          }
129          st[b] = 0;
130      }
131      inline bool on_found_edge(const edge &e){
132          int u = st[e.u], v = st[e.v];
133          if(S[v] == -1){
134              pa[v] = e.u, S[v] = 1;
135              int nu = st[match[v]];
```

```
136                    slack[v] = slack[nu] = 0;
137                    S[nu] = 0, q_push(nu);
138              }else if(S[v] == 0){
139                    int lca = get_lca(u, v);
140                    if(!lca) return augment(u, v), augment(v, u), true;
141                    else add_blossom(u, lca, v);
142              }
143          return false;
144      }
145      inline bool matching(){
146          memset(S + 1, -1, sizeof(int) * n_x);
147          memset(slack + 1, 0, sizeof(int) * n_x);
148          q = queue<int>();
149          for(int x = 1;x <= n_x; ++x)
150              if(st[x] == x && !match[x]) pa[x]=0, S[x]=0, q_push(x);
151          if(q.empty())return false;
152          for(;;){
153              while(q.size()){
154                  int u = q.front();q.pop();
155                  if(S[st[u]] == 1)continue;
156                  for(int v = 1;v <= n; ++v)
157                      if(g[u][v].w > 0 && st[u] != st[v]){
158                          if(e_delta(g[u][v]) == 0){
159                              if(on_found_edge(g[u][v]))return true;
160                          }else update_slack(u, st[v]);
161                      }
162              }
163              int d = INF;
164              for(int b = n + 1; b <= n_x;++b)
165                  if(st[b] == b && S[b] == 1)d = min(d, lab[b]/2);
166              for(int x = 1; x <= n_x; ++x)
167                  if(st[x] == x && slack[x]){
168                      if(S[x] == -1)d = min(d, e_delta(g[slack[x]][x]));
169                      else if(S[x] == 0)d = min(d, e_delta(g[slack[x]][x])/2);
170                  }
171              for(int u = 1; u <= n; ++u){
172                  if(S[st[u]] == 0){
173                      if(lab[u] <= d)return 0;
174                      lab[u] -= d;
175                  }else if(S[st[u]] == 1)lab[u] += d;
176              }
177              for(int b = n+1; b <= n_x; ++b)
178                  if(st[b] == b){
179                      if(S[st[b]] == 0) lab[b] += d * 2;
180                      else if(S[st[b]] == 1) lab[b] -= d * 2;
181                  }
```

```
182            q=queue<int>();
183            for(int x = 1; x <= n_x; ++x)
184                if(st[x] == x && slack[x] && st[slack[x]] != x && e_delta(g[slack[x]][x]) ==
    ↪ 0)
185                    if(on_found_edge(g[slack[x]][x]))return true;
186            for(int b = n + 1; b <= n_x; ++b)
187                if(st[b] == b && S[b] == 1 && lab[b] == 0)expand_blossom(b);
188        }
189        return false;
190    }
191    inline pair<long long, int> solve(){
192        memset(match + 1, 0, sizeof(int) * n);
193        n_x = n;
194        int n_matches = 0;
195        long long tot_weight = 0;
196        for(int u = 0; u <= n; ++u) st[u] = u, flower[u].clear();
197        int w_max = 0;
198        for(int u = 1; u <= n; ++u)
199            for(int v = 1; v <= n; ++v){
200                flower_from[u][v] = (u == v ? u : 0);
201                w_max = max(w_max, g[u][v].w);
202            }
203        for(int u = 1; u <= n; ++u) lab[u] = w_max;
204        while(matching()) ++n_matches;
205        for(int u = 1; u <= n; ++u)
206            if(match[u] && match[u] < u)
207                tot_weight += g[u][match[u]].w;
208        return make_pair(tot_weight, n_matches);
209    }
210    inline void init(){
211        for(int u = 1; u <= n; ++u)
212            for(int v = 1; v <= n; ++v)
213                g[u][v]=edge(u, v, 0);
214    }
215 };
```

无向图最小割

```
1 /*
2  * Stoer Wagner 全局最小割 O(V ^ 3)
3  * 1base, 点数 n, 邻接矩阵 edge[MAXN][MAXN]
4  * 返回值为全局最小割
5  */
6
7 int StoerWagner() {
```

```
 8        static int v[MAXN], wage[MAXN];
 9        static bool vis[MAXN];
10
11        for (int i = 1; i <= n; ++i) v[i] = i;
12
13        int res = INF;
14
15        for (int nn = n; nn > 1; --nn) {
16            memset(vis, 0, sizeof(bool) * (nn + 1));
17            memset(wage, 0, sizeof(int) * (nn + 1));
18
19            int pre, last = 1; // vis[1] = 1;
20
21            for (int i = 1; i < nn; ++i) {
22                pre = last; last = 0;
23                for (int j = 2; j <= nn; ++j) if (!vis[j]) {
24                    wage[j] += edge[v[pre]][v[j]];
25                    if (!last || wage[j] > wage[last]) last = j;
26                }
27                vis[last] = 1;
28            }
29
30            res = std::min(res, wage[last]);
31
32            for (int i = 1; i <= nn; ++i) {
33                edge[v[i]][v[pre]] += edge[v[last]][v[i]];
34                edge[v[pre]][v[i]] += edge[v[last]][v[i]];
35            }
36            v[last] = v[nn];
37        }
38        return res;
39    }
```

最大带权带花树

```
1  //maximum weight blossom,  change g[u][v].w to INF - g[u][v].w when minimum weight blossom
   ↪ is needed
2  //type of ans is long long
3  //replace all int to long long if weight of edge is long long
4
5  struct WeightGraph {
6      static const int INF = INT_MAX;
7      static const int MAXN = 400;
8      struct edge{
9          int u, v, w;
```

```cpp
        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)
```

```
56              set_match(flower[u][i], flower[u][i ^ 1]);
57          set_match(xr, v);
58          rotate(flower[u].begin(), flower[u].begin()+pr, flower[u].end());
59      }
60  }
61  inline void augment(int u, int v){
62      for(; ; ){
63          int xnv=st[match[u]];
64          set_match(u, v);
65          if(!xnv)return;
66          set_match(xnv, st[pa[xnv]]);
67          u=st[pa[xnv]], v=xnv;
68      }
69  }
70  inline int get_lca(int u, int v){
71      static int t=0;
72      for(++t; u || v; swap(u, v)){
73          if(u == 0)continue;
74          if(vis[u] == t)return u;
75          vis[u] = t;
76          u = st[match[u]];
77          if(u) u = st[pa[u]];
78      }
79      return 0;
80  }
81  inline void add_blossom(int u, int lca, int v){
82      int b = n + 1;
83      while(b <= n_x && st[b]) ++b;
84      if(b > n_x) ++n_x;
85      lab[b] = 0, S[b] = 0;
86      match[b] = match[lca];
87      flower[b].clear();
88      flower[b].push_back(lca);
89      for(int x = u, y; x != lca; x = st[pa[y]]) {
90          flower[b].push_back(x),
91          flower[b].push_back(y = st[match[x]]),
92          q_push(y);
93      }
94      reverse(flower[b].begin() + 1, flower[b].end());
95      for(int x = v, y; x != lca; x = st[pa[y]]) {
96          flower[b].push_back(x),
97          flower[b].push_back(y = st[match[x]]),
98          q_push(y);
99      }
100     set_st(b, b);
101     for(int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w = 0;
```

```
102        for(int x = 1; x <= n; ++x) flower_from[b][x] = 0;
103        for(size_t i = 0 ; i < flower[b].size(); ++i){
104            int xs = flower[b][i];
105            for(int x = 1; x <= n_x; ++x)
106                if(g[b][x].w == 0 || e_delta(g[xs][x]) < e_delta(g[b][x]))
107                    g[b][x] = g[xs][x], g[x][b] = g[x][xs];
108            for(int x = 1;x <= n; ++x)
109                if(flower_from[xs][x]) flower_from[b][x] = xs;
110        }
111        set_slack(b);
112    }
113    inline void expand_blossom(int b){ // S[b]  ==  1
114        for(size_t i = 0; i < flower[b].size(); ++i)
115            set_st(flower[b][i], flower[b][i]);
116        int xr = flower_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
117        for(int i = 0; i < pr; i += 2){
118            int xs = flower[b][i], xns = flower[b][i + 1];
119            pa[xs] = g[xns][xs].u;
120            S[xs] = 1, S[xns] = 0;
121            slack[xs] = 0, set_slack(xns);
122            q_push(xns);
123        }
124        S[xr] = 1, pa[xr] = pa[b];
125        for(size_t i = pr + 1;i < flower[b].size(); ++i){
126            int xs = flower[b][i];
127            S[xs] = -1, set_slack(xs);
128        }
129        st[b] = 0;
130    }
131    inline bool on_found_edge(const edge &e){
132        int u = st[e.u], v = st[e.v];
133        if(S[v] == -1){
134            pa[v] = e.u, S[v] = 1;
135            int nu = st[match[v]];
136            slack[v] = slack[nu] = 0;
137            S[nu] = 0, q_push(nu);
138        }else if(S[v] == 0){
139            int lca = get_lca(u, v);
140            if(!lca) return augment(u, v), augment(v, u), true;
141            else add_blossom(u, lca, v);
142        }
143        return false;
144    }
145    inline bool matching(){
146        memset(S + 1, -1, sizeof(int) * n_x);
147        memset(slack + 1, 0, sizeof(int) * n_x);
```

```
148          q = queue<int>();
149          for(int x = 1;x <= n_x; ++x)
150              if(st[x] == x && !match[x]) pa[x]=0, S[x]=0, q_push(x);
151          if(q.empty())return false;
152          for(;;){
153              while(q.size()){
154                  int u = q.front();q.pop();
155                  if(S[st[u]] == 1)continue;
156                  for(int v = 1;v <= n; ++v)
157                      if(g[u][v].w > 0 && st[u] != st[v]){
158                          if(e_delta(g[u][v]) == 0){
159                              if(on_found_edge(g[u][v]))return true;
160                          }else update_slack(u, st[v]);
161                      }
162              }
163              int d = INF;
164              for(int b = n + 1; b <= n_x;++b)
165                  if(st[b] == b && S[b] == 1)d = min(d, lab[b]/2);
166              for(int x = 1; x <= n_x; ++x)
167                  if(st[x] == x && slack[x]){
168                      if(S[x] == -1)d = min(d, e_delta(g[slack[x]][x]));
169                      else if(S[x] == 0)d = min(d, e_delta(g[slack[x]][x])/2);
170                  }
171              for(int u = 1; u <= n; ++u){
172                  if(S[st[u]] == 0){
173                      if(lab[u] <= d)return 0;
174                      lab[u] -= d;
175                  }else if(S[st[u]] == 1)lab[u] += d;
176              }
177              for(int b = n+1; b <= n_x; ++b)
178                  if(st[b] == b){
179                      if(S[st[b]] == 0) lab[b] += d * 2;
180                      else if(S[st[b]] == 1) lab[b] -= d * 2;
181                  }
182              q=queue<int>();
183              for(int x = 1; x <= n_x; ++x)
184                  if(st[x] == x && slack[x] && st[slack[x]] != x && e_delta(g[slack[x]][x]) ==
    ↪ 0)
185                      if(on_found_edge(g[slack[x]][x]))return true;
186              for(int b = n + 1; b <= n_x; ++b)
187                  if(st[b] == b && S[b] == 1 && lab[b] == 0)expand_blossom(b);
188          }
189          return false;
190      }
191      inline pair<long long, int> solve(){
192          memset(match + 1, 0, sizeof(int) * n);
```

```
193         n_x = n;
194         int n_matches = 0;
195         long long tot_weight = 0;
196         for(int u = 0; u <= n; ++u) st[u] = u, flower[u].clear();
197         int w_max = 0;
198         for(int u = 1; u <= n; ++u)
199             for(int v = 1; v <= n; ++v){
200                 flower_from[u][v] = (u == v ? u : 0);
201                 w_max = max(w_max, g[u][v].w);
202             }
203         for(int u = 1; u <= n; ++u) lab[u] = w_max;
204         while(matching()) ++n_matches;
205         for(int u = 1; u <= n; ++u)
206             if(match[u] && match[u] < u)
207                 tot_weight += g[u][match[u]].w;
208         return make_pair(tot_weight, n_matches);
209     }
210     inline void init(){
211         for(int u = 1; u <= n; ++u)
212             for(int v = 1; v <= n; ++v)
213                 g[u][v]=edge(u, v, 0);
214     }
215 };
```

必经点 Dominator-tree

```
1  //solve(s, n, raw_g): s is the root and base accords to base of raw_g
2  //idom[x] will be x if x does not have a dominator,and will be -1 if x is not reachable from
   ↪ s.
3
4  struct dominator_tree {
5      int base, dfn[N], sdom[N], idom[N], id[N], f[N], fa[N], smin[N], stamp;
6      Graph *g;
7      void predfs(int u) {
8          id[dfn[u] = stamp++] = u;
9          for (int i = g -> adj[u]; ~i; i = g -> nxt[i]) {
10             int v = g -> v[i];
11             if (dfn[v] < 0) {
12                 f[v] = u;
13                 predfs(v);
14             }
15         }
16     }
17     int getfa(int u) {
18         if (fa[u] == u) return u;
```

```
19              int ret = getfa(fa[u]);
20              if (dfn[sdom[smin[fa[u]]]] < dfn[sdom[smin[u]]])
21                  smin[u] = smin[fa[u]];
22              return fa[u] = ret;
23          }
24      void solve (int s, int n, Graph *raw_graph) {
25          g = raw_graph;
26          base = g -> base;
27          memset(dfn + base, -1, sizeof(*dfn) * n);
28          memset(idom + base, -1, sizeof(*idom) * n);
29          static Graph pred, tmp;
30          pred.init(base, n);
31          for (int i = 0; i < n; ++i) {
32              for (int p = g -> adj[i + base]; ~p; p = g -> nxt[p])
33                  pred.ins(g -> v[p], i + base);
34          }
35          stamp = 0; tmp.init(base, n); predfs(s);
36          for (int i = 0; i < stamp; ++i) {
37              fa[id[i]] = smin[id[i]] = id[i];
38          }
39          for (int o = stamp - 1; o >= 0; --o) {
40              int x = id[o];
41              if (o) {
42                  sdom[x] = f[x];
43                  for (int i = pred.adj[x]; ~i; i = pred.nxt[i]) {
44                      int p = pred.v[i];
45                      if (dfn[p] < 0) continue;
46                      if (dfn[p] > dfn[x]) {
47                          getfa(p);
48                          p = sdom[smin[p]];
49                      }
50                      if (dfn[sdom[x]] > dfn[p]) sdom[x] = p;
51                  }
52                  tmp.ins(sdom[x], x);
53              }
54              while (~tmp.adj[x]) {
55                  int y = tmp.v[tmp.adj[x]];
56                  tmp.adj[x] = tmp.nxt[tmp.adj[x]];
57                  getfa(y);
58                  if (x != sdom[smin[y]]) idom[y] = smin[y];
59                  else idom[y] = x;
60              }
61              for (int i = g -> adj[x]; ~i; i = g -> nxt[i])
62                  if (f[g -> v[i]] == x) fa[g -> v[i]] = x;
63          }
64          idom[s] = s;
```

```
65          for (int i = 1; i < stamp; ++i) {
66              int x = id[i];
67              if (idom[x] != sdom[x]) idom[x] = idom[idom[x]];
68          }
69      }
70  };
```

K 短路

```
1  //需保证 GivenEdge 里面边的顺序和 Edge 中一样
2  //两个优先队列要考虑大根还是小根
3  //heap 总是小根堆
4  //dij 不能求正权最长路
5  //INF or -INF
6
7  typedef long long LL;
8  MAXN, MAXK, MAXN, INF //int or LL, it depends
9  const int MAXNODE = MAXN + MAXM * 2;   // m + nlgm ???
10 bool used[MAXN];
11 int n, m, cnt, S, T, Kth, N;// m is number of all edges
12 int rt[MAXN], seq[MAXN], adj[MAXN], from[MAXN], dep[MAXN];
13 LL dist[MAXN], w[MAXM], ans[MAXK];
14
15 struct GivenEdge {  //edge given from origin input
16     int u, v, w;
17     GivenEdge() {};
18     GivenEdge(int _u, int _v, int _w): u(_u), v(_v), w(_w) {};
19 } edge[MAXM];
20
21 struct Edge {
22     int v, nxt, w;
23     Edge() {};
24     Edge(int _v, int _nxt, int _w): v(_v), nxt(_nxt), w(_w) {};
25 } e[MAXM];
26
27 inline void addedge(int u, int v, int w) {
28     e[++cnt] = Edge(v, adj[u], w); adj[u] = cnt;
29 }
30
31 inline void dij(int S) {  //dij in original graph, spfa if needed
32     for (int i = 1; i <= N; ++i) {
33         dist[i] = INF;
34         dep[i] = INF;
35         used[i] = false;
36         from[i] = 0;
```

```
37          }
38          static priority_queue<pair<LL, int>, vector<pair<LL, int> >, greater<pair<LL, int> > >
   ↪ hp;
39          while (!hp.empty()) hp.pop();
40          hp.push(make_pair(dist[S] = 0, S));
41          dep[S] = 1;
42          while (!hp.empty()) {
43              pair<LL, int> now = hp.top();
44              int u = now.second;
45              hp.pop();
46              if (used[u]) {
47                  continue;
48              } else {
49                  used[u] = true;
50              }
51              for (int p = adj[u]; p; p = e[p].nxt) {
52                  int v = e[p].v;
53                  if (dist[u] + e[p].w < dist[v]) {  //different when max or min
54                      dist[v] = dist[u] + e[p].w;
55                      dep[v] = dep[u] + 1;
56                      from[v] = p;
57                      hp.push(make_pair(dist[v], v));
58                  }
59              }
60          }
61          for (int i = 1; i <= m; ++i) w[i] = 0;
62          for (int i = 1; i <= N; ++i)
63              if (from[i]) w[from[i]] = -1;
64          for (int i = 1; i <= m; ++i) {
65              if (~w[i] && dist[edge[i].u] < INF && dist[edge[i].v] < INF) {
66                  w[i] = -dist[edge[i].u] + (dist[edge[i].v] + edge[i].w);     //different when max
   ↪ or min
67              } else {
68                  w[i] = -1;
69              }
70          }
71 }
72
73 inline bool cmp_dep(int p, int q) {
74      return dep[p] < dep[q];
75 }
76
77 struct Heap {
78      LL key;
79      int id, lc, rc, dist;
80      Heap() {};
```

```
 81        Heap(LL k, int i, int l, int r, int d): key(k), id(i), lc(l), rc(r), dist(d) {};
 82        inline void clear() {
 83            key = 0;
 84            id = lc = rc = dist = 0;
 85        }
 86  } hp[MAXNODE];
 87
 88  inline int merge_simple(int u, int v) {
 89        if (!u) return v;
 90        if (!v) return u;
 91        if (hp[u].key > hp[v].key) {
 92            swap(u, v);
 93        }
 94        hp[u].rc = merge_simple(hp[u].rc, v);
 95        if (hp[hp[u].lc].dist < hp[hp[u].rc].dist) {
 96            swap(hp[u].lc, hp[u].rc);
 97        }
 98        hp[u].dist = hp[hp[u].rc].dist + 1;
 99        return u;
100  }
101
102  inline int merge_full(int u, int v) {
103        if (!u) return v;
104        if (!v) return u;
105        if (hp[u].key > hp[v].key) {
106            swap(u, v);
107        }
108        int nownode = ++cnt;
109        hp[nownode] = hp[u];
110        hp[nownode].rc = merge_full(hp[nownode].rc, v);
111        if (hp[hp[nownode].lc].dist < hp[hp[nownode].rc].dist) {
112            swap(hp[nownode].lc, hp[nownode].rc);
113        }
114        hp[nownode].dist = hp[hp[nownode].rc].dist + 1;
115        return nownode;
116  }
117
118  priority_queue<pair<LL, int>, vector<pair<LL, int> >, greater<pair<LL, int> > > Q;
119
120  int main() {
121        scanf("%d%d%d", &n, &m, &Kth);
122        for (int i = 1; i <= m; ++i) {
123            int u, v, w;
124            scanf("%d%d%d", &u, &v, &w);
125            edge[i] = {u, v, w};
126        }
```

```
127        N = ;
128        S = ;
129        T = ;
130        memset(adj, 0, sizeof(*adj) * (N + 1));
131        cnt = 0;
132        for (int i = 1; i <= m; ++i) {
133            addedge(edge[i].v, edge[i].u, edge[i].w); // important!!! reverse the edge
134        }
135        dij(T);
136        if (dist[S] == INF) {    //must judge before building heaps; -INF if max kth
137            ...
138            return 0;
139        }
140        for (int i = 1; i <= N; ++i) {
141            seq[i] = i;
142        }
143        sort(seq + 1, seq + N + 1, cmp_dep);
144
145        cnt = 0;
146        memset(adj, 0, sizeof(*adj) * (N + 1));
147        memset(rt, 0, sizeof(*rt) * (N + 1));
148        for (int i = 1; i <= m; ++i) {
149            addedge(edge[i].u, edge[i].v, edge[i].w);
150        }
151        rt[T] = cnt = 0;  // now cnt is total nodes in heaps
152        hp[0].dist = -1;
153        for (int i = 1; i <= N; ++i) {
154            int u = seq[i], v = edge[from[u]].v;
155            rt[u] = 0;
156            for (int p = adj[u]; p; p = e[p].nxt) {
157                if (~w[p]) {
158                    hp[++cnt] = Heap(w[p], p, 0, 0, 0);
159                    rt[u] = merge_simple(rt[u], cnt);
160                }
161            }
162            if (i == 1) continue;
163            rt[u] = merge_full(rt[u], rt[v]);
164        }
165        while (!Q.empty()) Q.pop();
166        Q.push(make_pair(dist[S], 0));
167        edge[0].v = S;
168        for (int kth = 1; kth <= Kth; ++kth) {
169            if (Q.empty()) {
170                ans[kth] = -1;
171                continue;
172            }
```

```
173        pair<LL, int> now = Q.top(); Q.pop();
174        ans[kth] = now.first;
175        int p = now.second;
176        if (hp[p].lc) {
177            Q.push(make_pair(+hp[hp[p].lc].key + now.first - hp[p].key,
    ↪hp[p].lc));//different when max or min
178        }
179        if (hp[p].rc) {
180            Q.push(make_pair(+hp[hp[p].rc].key + now.first - hp[p].key,
    ↪hp[p].rc));//different when max or min
181        }
182        if (rt[edge[hp[p].id].v]) {
183            Q.push(make_pair(hp[rt[edge[hp[p].id].v]].key + now.first,
    ↪rt[edge[hp[p].id].v]));//different when max or min
184        }
185    }
186    ...
187    for (int i = 1; i <= cnt; ++i) {
188        hp[i].clear();
189    }
190 }
```

最大团搜索

```
1  Int g[][]为图的邻接矩阵。
2      MC(V)表示点集V的最大团
3      令Si={vi, vi+1, ..., vn}, mc[i]表示MC(Si)
4      倒着算mc[i]，那么显然MC(V)=mc[1]
5      此外有mc[i]=mc[i+1] or mc[i]=mc[i+1]+1
6  void init(){
7      int i, j;
8      for (i=1; i<=n; ++i) for (j=1; j<=n; ++j) scanf("%d", &g[i][j]);
9  }
10 void dfs(int size){
11     int i, j, k;
12     if (len[size]==0) {
13         if (size>ans) {
14             ans=size; found=true;
15         }
16         return;
17     }
18     for (k=0; k<len[size] && !found; ++k) {
19         if (size+len[size]-k<=ans) break;
20         i=list[size][k];
21         if (size+mc[i]<=ans) break;
```

```
22        for (j=k+1, len[size+1]=0; j<len[size]; ++j)
23            if (g[i][list[size][j]]) list[size+1][len[size+1]++]=list[size][j];
24        dfs(size+1);
25    }
26 }
27 void work(){
28    int i, j;
29    mc[n]=ans=1;
30    for (i=n-1; i; --i) {
31        found=false;
32        len[1]=0;
33        for (j=i+1; j<=n; ++j) if (g[i][j]) list[1][len[1]++]=j;
34        dfs(1);
35        mc[i]=ans;
36    }
37 }
38 void print(){
39    printf("%d\n", ans);
40 }
```

极大团计数

```
1 Bool g[][]为图的邻接矩阵，图点的标号由1至n。
2 void dfs(int size){
3    int i, j, k, t, cnt, best = 0;
4    bool bb;
5    if (ne[size]==ce[size]){
6        if (ce[size]==0) ++ans;
7        return;
8    }
9    for (t=0, i=1; i<=ne[size]; ++i) {
10        for (cnt=0, j=ne[size]+1; j<=ce[size]; ++j)
11        if (!g[list[size][i]][list[size][j]]) ++cnt;
12        if (t==0 || cnt<best) t=i, best=cnt;
13    }
14    if (t && best<=0) return;
15    for (k=ne[size]+1; k<=ce[size]; ++k) {
16        if (t>0){
17            for (i=k; i<=ce[size]; ++i) if (!g[list[size][t]][list[size][i]]) break;
18            swap(list[size][k], list[size][i]);
19        }
20        i=list[size][k];
21        ne[size+1]=ce[size+1]=0;
22        for (j=1; j<k; ++j)if (g[i][list[size][j]])
   ↪ list[size+1][++ne[size+1]]=list[size][j];
```

```
23        for (ce[size+1]=ne[size+1], j=k+1; j<=ce[size]; ++j)
24            if (g[i][list[size][j]]) list[size+1][++ce[size+1]]=list[size][j];
25        dfs(size+1);
26        ++ne[size];
27        --best;
28        for (j=k+1, cnt=0; j<=ce[size]; ++j) if (!g[i][list[size][j]]) ++cnt;
29        if (t==0 || cnt<best) t=k, best=cnt;
30        if (t && best<=0) break;
31    }
32 }
33 void work(){
34    int i;
35    ne[0]=0; ce[0]=0;
36    for (i=1; i<=n; ++i) list[0][++ce[0]]=i;
37    ans=0;
38    dfs(0);
39 }
```

欧拉回路

```
1  //从一个奇度点 dfs，sqn 即为回路/路径
2  //first 存点，second 存边的编号，正反边编号一致
3  //清空 cur、used 数组
4  void getCycle(int u)
5  {
6      for(int &i=cur[u]; i < (int)adj[u].size(); ++ i) {
7          int id = adj[u][i].second;
8          if (used[id]) continue;
9          used[id] = true;
10         getCycle(adj[u][i].first);
11     }
12     sqn.push_back(u);
13 }
```

朱刘最小树形图

```
1  struct D_MT {
2      struct Edge {
3          int u, v, w;
4          inline Edge() {}
5          inline Edge(int _u, int _v, int _w):u(_u), v(_v), w(_w) {
6          }
7      };
8      int nn, mm, n, m, vis[maxn], pre[maxn], id[maxn], in[maxn];
```

```
 9    Edge edges[maxn], bac[maxn];
10    void init(int _n) {
11        n = _n;
12        m = 0;
13    }
14    void AddEdge(int u, int v, int w) {
15        edges[m++] = Edge(u, v, w);
16    }
17    int work(int root) {
18        int ret = 0;
19        while(true) {
20            for (int i = 0; i < n; i++) in[i]=inf + 1;
21            for (int i = 0; i < m; i++) {
22                int u = edges[i].u, v = edges[i].v;
23                if(edges[i].w < in[v] && u != v){
24                    in[v] = edges[i].w;
25                    pre[v] = u;
26                }
27            }
28            for (int i = 0; i < n; i++) {
29                if(i == root) continue;
30                if(in[i] == inf + 1) return inf;
31            }
32            int cnt = 0;
33            for (int i = 0; i < n; i++) {
34                id[i] = -1;
35                vis[i] = -1;
36            }
37            in[root] = 0;
38            for (int i = 0; i < n; i++) {
39                ret += in[i];
40                int v = i;
41                while (vis[v] != i&& id[v] == -1 && v != root ){
42                    vis[v] = i;
43                    v = pre[v];
44                }
45                if (v != root && id[v] == -1) {
46                    for (int u = pre[v]; u != v; u = pre[u]) id[u] = cnt;
47                    id[v] = cnt++;
48                }
49            }
50            if (!cnt) break;
51            for (int i=0; i<n; i++)
52                if (id[i] == -1) id[i] = cnt++;
53            for (int i = 0; i < m; i++){
54                int u = edges[i].u, v = edges[i].v;
```

```
55              edges[i].v = id[v];
56              edges[i].u = id[u];
57              if(id[u] != id[v]) edges[i].w -= in[v];
58          }
59          n = cnt;
60          root = id[root];
61      }
62      return ret;
63    }
64 } MT;
```

Chapter 4

数据结构

Kd-tree

```
1  int n;
2  LL norm(const LL &x) {
3      //    For manhattan distance
4      //return std::abs(x);
5      //    For euclid distance
6      return x * x;
7  }
8
9  struct P{
10     int a[2],val;
11     int id;
12     int& operator[](int s){return a[s];}
13     const int& operator[](int s)const{return a[s];}
14
15     LL dis(const P &b)const{
16         LL ans=0;
17         for (int i = 0; i < 2; ++i) {
18             ans += norm(a[i] - b[i]);
19         }
20         return ans;
21     }
22 }p[maxn];
23
24 bool operator==(const P &a,const P &b){
25     for(int i=0;i<DIM;i++)
26         if(a[i]!=b[i])
27             return false;
28     return true;
29 }
30 bool byVal(P a,P b){
31     return a.val!=b.val ? a.val<b.val : a.id<b.id;
```

```
32  }
33
34  struct Rec{
35      int mn[DIM],mx[DIM];
36      Rec(){}
37      Rec(const P &p){
38          for(int i=0;i<DIM;i++){
39              mn[i]=mx[i]=p[i];
40          }
41      }
42      void add(const P &p){
43          for(int i=0;i<DIM;i++){
44              mn[i]=min(p[i],mn[i]);
45              mx[i]=max(p[i],mx[i]);
46          }
47      }
48
49      LL dis(const P &p) {
50          LL ans = 0;
51          for (int i = 0; i < 2; ++i) {
52              //    For minimum distance
53              ans += norm(min(max(p[i], mn[i]), mx[i]) - p[i]);
54              //    For maximum distance
55              //ans += std::max(norm(max[i] - p[i]), norm(min[i] - p[i]));
56          }
57          return ans;
58      }
59  };
60  inline Rec operator+(const Rec &ls,const Rec &rs){
61      static Rec rec;
62      for(int i=0;i<DIM;i++){
63          rec.mn[i]=min(ls.mn[i],rs.mn[i]);
64          rec.mx[i]=max(ls.mx[i],rs.mx[i]);
65      }
66      return rec;
67  }
68  struct node{
69      Rec rec;
70      P sep;
71      int sum,siz;
72      node *c[2];
73      node *rz(){
74          sum=sep.val;
75          rec=Rec(sep);
76          siz=1;
77          if(c[0]){
```

```
78              sum+=c[0]->sum;
79              rec=rec+c[0]->rec;
80              siz+=c[0]->siz;
81          }
82          if(c[1]){
83              sum+=c[1]->sum;
84              rec=rec+c[1]->rec;
85              siz+=c[1]->siz;
86          }
87          return this;
88      }
89      node(){sum=0;siz=1;c[0]=c[1]=0;}
90 }*root,*re,pool[maxn],*cur=pool;
91 node *sta[maxn];
92 P tmp[maxn];
93 int D,si;
94 void init(){
95      si=0;
96      cur=pool;
97      root=0;
98 }
99 bool cmp(const P &A,const P &B){
100
101      if(!(A[D]==B[D]))
102          return A[D]<B[D];
103      return A.id<B.id;
104 }
105 int top;
106 node *newnode(){
107      if(si)return sta[si--];
108      return cur++;
109 }
110 node* build(P *p,int l,int r,int d){
111      int mid=(l+r)>>1;D=d;
112      nth_element(p+l,p+mid,p+r+1,cmp);
113      node *t=newnode();
114      t->sep=p[mid];
115      if(l<=mid-1)
116          t->c[0]=build(p,l,mid-1,d^1);
117      if(mid+1<=r)
118          t->c[1]=build(p,mid+1,r,d^1);
119      return t->rz();
120 }
121 void dfs(node *&t){
122      if(t->c[0])dfs(t->c[0]);
123      tmp[++top]=t->sep;
```

```
124        if(t->c[1])dfs(t->c[1]);
125        sta[++si]=t;*t=node();
126        //delete t;
127    }
128    node* rebuild(node *&t){
129        if(!t)return 0;
130        top=0;dfs(t);
131        return build(tmp,1,top,0);
132    }
133    #define siz(x) (x?x->siz:0)
134    void Add(node *&t,const P &p,int d=0){//调用前 re=0; 调用后 rebuild(re);
135        D=d;
136        if(!t){
137            t=newnode();
138            t->sep=p;t->rz();
139            return;
140        }
141        if(t->sep==p){
142            t->sep.val+=p.val;
143            t->rz();
144            return;
145        }
146        if(p[D]<t->sep[D])
147            Add(t->c[0],p,d^1);
148        else
149            Add(t->c[1],p,d^1);
150
151        t->rz();
152
153        if(max(siz(t->c[0]),siz(t->c[1]))>0.7*t->siz)
154            re=t;
155    }
156    int ans;
157
158    bool Out(const Rec &a,const Rec &b){
159        for(int i=0;i<DIM;i++){
160            int l=max(a.mn[i],b.mn[i]);
161            int r=min(a.mx[i],b.mx[i]);
162            if(l>r)
163                return true;
164        }
165        return false;
166    }
167    bool In(const Rec &a,const Rec &b){
168        for(int i=0;i<DIM;i++){
169            if(a.mn[i]<b.mn[i])
```

```
170                 return false;
171             if(a.mx[i]>b.mx[i])
172                 return false;
173         }
174         return true;
175 }
176
177 bool In(const P &a,const Rec &b){
178         for(int i=0;i<DIM;i++){
179             if(!(b.mn[i]<=a[i]&&a[i]<=b.mx[i]))
180                 return false;
181         }
182         return true;
183 }
184
185 void Q(node *t,const Rec &R){
186         if(Out(t->rec,R))return ;
187         if(In(t->rec,R)){
188             ans+=t->sum;
189             return;
190         }
191         if(In(t->sep,R))
192             ans+=t->sep.val;
193         if(t->c[0])
194             Q(t->c[0],R);
195         if(t->c[1])
196             Q(t->c[1],R);
197 }
198
199 priority_queue<pair<long long, int> > kNN;
200 void query(node *t, const P &p, int k, int d = 0) {//用钱清空 kNN
201         D=d;
202         if (!t || ((int)kNN.size() == k && t->rec.dis(p) > kNN.top().first)) {
203             return;
204         }
205         kNN.push(make_pair(t->sep.dis(p), t->sep.id));
206         if ((int)kNN.size() > k) {
207             kNN.pop();
208         }
209         if (cmp(p, t->sep)) {
210             query(t->c[0], p, k, d ^ 1);
211             query(t->c[1], p, k, d ^ 1);
212         } else {
213             query(t->c[1], p, k, d ^ 1);
214             query(t->c[0], p, k, d ^ 1);
215         }
```

```
216  }
```

LCT

```
1   struct LCT{
2       struct node{
3           bool rev;
4           int mx,val;
5           node *f,*c[2];
6           bool d(){return this==f->c[1];}
7           bool rt(){return !f||(f->c[0]!=this&&f->c[1]!=this);}
8           void sets(node *x,int d){pd();if(x)x->f=this;c[d]=x;rz();}
9           void makerv(){rev^=1;swap(c[0],c[1]);}
10          void pd(){
11              if(rev){
12                  if(c[0])c[0]->makerv();
13                  if(c[1])c[1]->makerv();
14                  rev=0;
15              }
16          }
17          void rz(){
18              mx=val;
19              if(c[0])mx=max(mx,c[0]->mx);
20              if(c[1])mx=max(mx,c[1]->mx);
21          }
22      }nd[int(1e4)+1];
23      void rot(node *x){
24          node *y=x->f;if(!y->rt())y->f->pd();
25          y->pd();x->pd();bool d=x->d();
26          y->sets(x->c[!d],d);
27          if(y->rt())x->f=y->f;
28          else y->f->sets(x,y->d());
29          x->sets(y,!d);
30      }
31      void splay(node *x){
32          while(!x->rt())
33              if(x->f->rt())rot(x);
34              else if(x->d()==x->f->d())rot(x->f),rot(x);
35              else rot(x),rot(x);
36      }
37      node* access(node *x){
38          node *y=0;
39          for(;x;x=x->f){
40              splay(x);
41              x->sets(y,1);y=x;
```

```
42          }return y;
43      }
44      void makert(node *x){
45          access(x)->makerv();
46          splay(x);
47      }
48      void link(node *x,node *y){
49          makert(x);
50          x->f=y;
51          access(x);
52      }
53      void cut(node *x,node *y){
54          makert(x);access(y);splay(y);
55          y->c[0]=x->f=0;
56          y->rz();
57      }
58      void link(int x,int y){link(nd+x,nd+y);}
59      void cut(int x,int y){cut(nd+x,nd+y);}
60  }T;
```

树状数组上二分第 k 大

```
1  int find(int k){
2      int cnt=0,ans=0;
3      for(int i=22;i>=0;i--){
4          ans+=(1<<i);
5          if(ans>n || cnt+d[ans]>=k)ans-=(1<<i);
6          else cnt+=d[ans];
7      }
8      return ans+1;
9  }
```

Treap

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

```
11          val=v;
12          key=rand();
13          siz=1,s=1;
14          c[0]=c[1]=0;
15      }
16      void rz(){siz=s;if(c[0])siz+=c[0]->siz;if(c[1])siz+=c[1]->siz;}
17   }pool[maxn],*cur,*root;
18   Treap(){cur=pool;}
19   node* newnode(int val){return *cur=node(val),cur++;}
20   void rot(node *&t,int d){
21       if(!t->c[d])t=t->c[!d];
22       else{
23           node *p=t->c[d];t->c[d]=p->c[!d];
24           p->c[!d]=t;t->rz();p->rz();t=p;
25       }
26   }
27   void insert(node *&t,int x){
28       if(!t){t=newnode(x);return;}
29       if(t->val==x){t->s++;t->siz++;return;}
30       insert(t->c[x>t->val],x);
31       if(t->key<t->c[x>t->val]->key)
32           rot(t,x>t->val);
33       else t->rz();
34   }
35   void del(node *&t,int x){
36       if(!t)return;
37       if(t->val==x){
38           if(t->s>1){t->s--;t->siz--;return;}
39           if(!t->c[0]||!t->c[1]){
40               if(!t->c[0])t=t->c[1];
41               else t=t->c[0];
42               return;
43           }
44           int d=t->c[0]->key<t->c[1]->key;
45           rot(t,d);
46           del(t,x);
47           return;
48       }
49       del(t->c[x>t->val],x);
50       t->rz();
51   }
52   int pre(node *t,int x){
53       if(!t)return INT_MIN;
54       int ans=pre(t->c[x>t->val],x);
55       if(t->val<x)ans=max(ans,t->val);
56       return ans;
```

```
57        }
58        int nxt(node *t,int x){
59            if(!t)return INT_MAX;
60            int ans=nxt(t->c[x>=t->val],x);
61            if(t->val>x)ans=min(ans,t->val);
62            return ans;
63        }
64        int rank(node *t,int x){
65            if(!t)return 0;
66            if(t->val==x)return sz(t->c[0]);
67            if(t->val<x)return sz(t->c[0])+t->s+rank(t->c[1],x);
68            if(t->val>x)return rank(t->c[0],x);
69        }
70        int kth(node *t,int x){
71            if(sz(t->c[0])>=x)return kth(t->c[0],x);
72            if(sz(t->c[0])+t->s>=x)return t->val;
73            return kth(t->c[1],x-t->s-sz(t->c[0]));
74        }
75        void deb(node *t){
76            if(!t)return;
77            deb(t->c[0]);
78            printf("%d ",t->val);
79            deb(t->c[1]);
80        }
81        void insert(int x){insert(root,x);}
82        void del(int x){del(root,x);}
83        int pre(int x){return pre(root,x);}
84        int nxt(int x){return nxt(root,x);}
85        int rank(int x){return rank(root,x);}
86        int kth(int x){return kth(root,x);}
87        void deb(){deb(root);puts("");}
88 }T;
```

FHQ-Treap

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

```cpp
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{
```

```
57          c[1]->split(ned-sz(c[0])-1,p,q);c[1]=0;rz();
58          p=merge(this,p);
59      }
60 }
61 int tot;
62 void C(int l,int r,int v){
63      node *p,*q,*x,*y;
64      root->split(l-1,p,q);
65      q->split(r-l+1,x,y);
66      x->make(0,v,0);x->pd();
67      root=merge(p,merge(x,y));
68 }
69 void A(int l,int r,int d){
70      node *p,*q,*x,*y;
71      root->split(l-1,p,q);
72      q->split(r-l+1,x,y);
73      x->make(1,d,d);x->pd();
74      root=merge(p,merge(x,y));
75 }
76 void I(int ps,int v){
77      node *p,*q;
78      root->split(ps-1,p,q);
79      node *x=nd+(++tot);
80      x->key=rand();x->val=v;x->rz();
81      root=merge(merge(p,x),q);
82 }
83 LL Q(int l,int r){
84      node *p,*q,*x,*y;
85      root->split(l-1,p,q);
86      q->split(r-l+1,x,y);
87      LL ans=x->sum;
88      root=merge(p,merge(x,y));
89      return ans;
90 }
91 int main(){
92 //  freopen("bzoj3188.in","r",stdin);
93      n=in();m=in();
94      for(int i=1;i<=n;i++){
95          nd[i].val=in();
96          nd[i].key=rand();
97          nd[i].rz();
98          root=merge(root,nd+i);
99      }tot=n;
100     while(m--){
101         int ty=in();
102         int l,r;
```

```
103        if(ty==1){
104            l=in();r=in();
105            C(l,r,in());
106        }else if(ty==2){
107            l=in();r=in();
108            A(l,r,in());
109        }else if(ty==3){
110            int ps=in();
111            I(ps,in());
112        }else if(ty==4){
113            l=in();r=in();
114            printf("%lld\n",Q(l,r));
115        }
116    }
117    return 0;
118 }
```

真-FHQTreap

```
1  const int mo=1e9+7;
2  int rnd(){
3      static int x=1;
4      return x=(x*23333+233);
5  }
6  int rnd(int n){
7      int x=rnd();
8      if(x<0)x=-x;
9      return x%n+1;
10 }
11 struct node{
12     int siz,key;
13     int val;
14     LL sum;
15     node *c[2];
16     node* rz(){
17         sum=val;siz=1;
18         if(c[0])sum+=c[0]->sum,siz+=c[0]->siz;
19         if(c[1])sum+=c[1]->sum,siz+=c[1]->siz;
20         return this;
21     }
22     node(){}
23     node(int v){
24         siz=1;key=rnd();
25         val=v;sum=v;
26         c[0]=c[1]=0;
```

```
27        }
28
29 }pool[maxn*8],*root,*cur=pool,*old_root,*stop;
30 node *newnode(int v=0){
31     *cur=node(v);
32     return cur++;
33 }
34 node *old_merge(node *p,node *q){
35     if(!p&&!q)return 0;
36     node *u=0;
37     if(!p||!q)return u=p?p->rz():(q?q->rz():0);
38     if(rnd(sz(p)+sz(q))<sz(p)){
39         u=p;
40         u->c[1]=old_merge(u->c[1],q);
41     }else{
42         u=q;
43         u->c[0]=old_merge(p,u->c[0]);
44     }
45     return u->rz();
46 }
47 node *merge(node *p,node *q){
48     if(!p&&!q)return 0;
49     node *u=newnode();
50     if(!p||!q)return u=p?p->rz():(q?q->rz():0);
51     if(rnd(sz(p)+sz(q))<sz(p)){
52         *u=*p;
53         u->c[1]=merge(u->c[1],q);
54     }else{
55         *u=*q;
56         u->c[0]=merge(p,u->c[0]);
57     }
58     return u->rz();
59 }
60 node *split(node *u,int l,int r){
61     if(l>r||!u)return 0;
62     node *x=0;
63     if(l==1&&r==sz(u)){
64         x=newnode();
65         *x=*u;
66         return x->rz();
67     }
68     int lsz=sz(u->c[0]);
69     if(r<=lsz)
70         return split(u->c[0],l,r);
71     if(l>lsz+1)
72         return split(u->c[1],l-lsz-1,r-lsz-1);
```

```
73      x=newnode();
74      *x=*u;
75      x->c[0]=split(u->c[0],l,lsz);
76      x->c[1]=split(u->c[1],1,r-lsz-1);
77      return x->rz();
78  }
```

莫队上树

```
1  bool operator<(qes a,qes b){
2      if(dfn[a.x]/B!=dfn[b.x]/B)return dfn[a.x]/B<dfn[b.x]/B;
3      if(dfn[a.y]/B!=dfn[b.y]/B)return dfn[a.y]/B<dfn[b.y]/B;
4      if(a.tm/B!=b.tm/B)return a.tm/B<b.tm/B;
5      return a.tm<b.tm;
6  }
7  void vxor(int x){
8      if(vis[x])ans-=(LL)W[cnt[col[x]]]*V[col[x]],cnt[col[x]]--;
9      else cnt[col[x]]++,ans+=(LL)W[cnt[col[x]]]*V[col[x]];
10     vis[x]^=1;
11 }
12 void change(int x,int y){
13     if(vis[x]){
14         vxor(x);col[x]=y;vxor(x);
15     }else col[x]=y;
16 }
17 void TimeMachine(int tar){//XD
18     for(int i=now+1;i<=tar;i++)change(C[i].x,C[i].y);
19     for(int i=now;i>tar;i--)change(C[i].x,C[i].pre);
20     now=tar;
21 }
22 void vxor(int x,int y){
23     while(x!=y)if(dep[x]>dep[y])vxor(x),x=fa[x];
24     else vxor(y),y=fa[y];
25 }
26     for(int i=1;i<=q;i++){
27         int ty=getint(),x=getint(),y=getint();
28         if(ty&&dfn[x]>dfn[y])swap(x,y);
29         if(ty==0) C[++Csize]=(oper){x,y,pre[x],i},pre[x]=y;
30         else Q[Qsize+1]=(qes){x,y,Qsize+1,Csize},Qsize++;
31     }sort(Q+1,Q+1+Qsize);
32     int u=Q[1].x,v=Q[1].y;
33     TimeMachine(Q[1].tm);
34     vxor(Q[1].x,Q[1].y);
35     int LCA=lca(Q[1].x,Q[1].y);
36     vxor(LCA);anss[Q[1].id]=ans;vxor(LCA);
```

```
37      for(int i=2;i<=Qsize;i++){
38          TimeMachine(Q[i].tm);
39          vxor(Q[i-1].x,Q[i].x);
40          vxor(Q[i-1].y,Q[i].y);
41          int LCA=lca(Q[i].x,Q[i].y);
42          vxor(LCA);
43          anss[Q[i].id]=ans;
44          vxor(LCA);
45      }
```

虚树

```
1  int a[maxn*2],sta[maxn*2];
2  int top=0,k;
3  void build(){
4      top=0;
5      sort(a,a+k,bydfn);
6      k=unique(a,a+k)-a;
7      sta[top++]=1;_n=k;
8      for(int i=0;i<k;i++){
9          int LCA=lca(a[i],sta[top-1]);
10         while(dep[LCA]<dep[sta[top-1]]){
11             if(dep[LCA]>=dep[sta[top-2]]){
12                 add_edge(LCA,sta[--top]);
13                 if(sta[top-1]!=LCA)sta[top++]=LCA;
14                 break;
15             }add_edge(sta[top-2],sta[top-1]);top--;
16         }if(sta[top-1]!=a[i])sta[top++]=a[i];
17     }
18     while(top>1)
19         add_edge(sta[top-2],sta[top-1]),top--;
20     for(int i=0;i<k;i++)inr[a[i]]=1;
21 }
```

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!
 */

```

```
 6  namespace PAM {
 7      struct Node *pool_pointer;
 8      struct Node {
 9          Node *fail, *to[26];
10          int cnt, len;
11
12          Node() {}
13          Node(int len): len(len) {
14              memset(to, 0, sizeof(to));
15              fail = 0;
16              cnt = 0;
17          }
18
19          void *operator new (size_t) {
20              return pool_pointer++;
21          }
22      } pool[100005], *root[2], *last;
23      int pam_len, str[100005];
24
25      void init() {
26          pool_pointer = pool;
27          root[0] = new Node(0);
28          root[1] = new Node(-1);
29          root[0]->fail = root[1]->fail = root[1];
30          str[pam_len = 0] = -1; // different from all characters
31          last = root[0];
32      }
33
34      void extend(char ch) {
35          static Node *p, *np, *q;
36
37          int x = str[++pam_len] = ch - 'a';
38
39          p = last;
40          while (str[pam_len - p->len - 1] != x)
41              p = p->fail;
42          if (!p->to[x]) {
43              np = new Node(p->len + 2), q = p->fail;
44              while (str[pam_len - q->len - 1] != x) q = q->fail;
45              np->fail = q->to[x] ? q->to[x] : root[0];
46              p->to[x] = np;
47          }
48          last = p->to[x];
49          ++last->cnt;
50      }
51  }
```

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

```
1  /*
2   * Suffix Automaton - pointer version
3   * SAMSAMSAM? SAMSAMSAM!
4   */
5
6  namespace SAM {
7      struct Node *pool_pointer;
8      struct Node {
9          Node *to[26], *parent;
10         int step;
11
12         Node(int STEP = 0): step(STEP) {
13             memset(to, 0, sizeof to);
14             parent = 0;
15             step = 0;
16         }
17
18         void *operator new (size_t) {
19             return pool_pointer++;
20         }
21     } pool[100005 << 1], *root, *np;
22     int sam_len;
23
24     void init() {
25         pool_pointer = pool;
26         root = np = new Node(sam_len = 0);
27     }
28
29     void extend(char ch) {
30         static Node *last, *q, *nq;
31
32         int x = ch - 'a';
33         last = np; np = new Node(++sam_len);
34         for (; last && !last->to[x]; last = last->parent)
35             last->to[x] = np;
36         if (!last) np->parent = root;
37         else {
38             q = last->to[x];
39             if (q->step == last->step + 1) np->parent = q;
40             else {
41                 nq = new Node(*q);
42                 nq->step = last->step + 1;
43                 q->parent = np->parent = nq;
44                 for (; last && last->to[x] == q; last = last->parent)
```

```
45                      last->to[x] = nq;
46                  }
47              }
48          }
49  }
```

广义后缀自动机

```
1   /*
2    * EX Suffix Automaton - pointer version
3    * SAMSAMSAM? SAMSAMSAM!
4    */
5
6   namespace SAM {
7       struct Node *pool_pointer;
8       struct Node {
9           Node *parent, *to[26];
10          int step;
11
12          Node(int step = 0): step(step) {
13              memset(to, 0, sizeof to);
14              parent = 0;
15          }
16
17          void *operator new (size_t) {
18              return pool_pointer++;
19          }
20      } pool[100005 * 10 << 1], *root, *np;
21      int sam_len, now_len;
22
23      void init() {
24          sam_len = now_len = 0;
25          pool_pointer = pool;
26          root = new Node();
27      }
28
29      void new_str() { // a new string start
30          now_len = 0;
31          np = root;
32      }
33
34      void extend(char ch) {
35          static Node *last, *q, *nq;
36
37          int x = ch - 'a';
```

```
38          if (np->to[x]) {
39              np = np->to[x];
40              ++now_len;
41          }
42          else {
43              last = np; np = new Node(++now_len);
44              for (; last && !last->to[x]; last = last->parent)
45                  last->to[x] = np;
46              if (!last) np->parent = root;
47              else {
48                  q = last->to[x];
49                  if (q->step == last->step + 1) np->parent = q;
50                  else {
51                      nq = new Node(*q);
52                      nq->step = last->step + 1;
53                      q->parent = np->parent = nq;
54                      for (; last && last->to[x] == q; last = last->parent)
55                          last->to[x] = nq;
56                  }
57              }
58          }
59
60          sam_len = std::max(sam_len, now_len);
61      }
62 }
```

后缀数组

```
1  const int maxl=1e5+1e4+5;
2  const int maxn=maxl*2;
3  int a[maxn],x[maxn],y[maxn],c[maxn],sa[maxn],rank[maxn],height[maxn];
4  void calc_sa(int n){
5      int m=alphabet,k=1;
6      memset(c,0,sizeof(*c)*(m+1));
7      for(int i=1;i<=n;i++)c[x[i]=a[i]]++;
8      for(int i=1;i<=m;i++)c[i]+=c[i-1];
9      for(int i=1;i<=n;i++)sa[c[x[i]]--]=i;
10     for(;k<=n;k<<=1){
11         int tot=k;
12         for(int i=n-k+1;i<=n;i++)y[i-n+k]=i;
13         for(int i=1;i<=n;i++)
14             if(sa[i]>k)y[++tot]=sa[i]-k;
15         memset(c,0,sizeof(*c)*(m+1));
16         for(int i=1;i<=n;i++)c[x[i]]++;
17         for(int i=1;i<=m;i++)c[i]+=c[i-1];
```

```
18        for(int i=n;i>=1;i--)sa[c[x[y[i]]]--]=y[i];
19        for(int i=1;i<=n;i++)y[i]=x[i];
20        tot=1;x[sa[1]]=1;
21        for(int i=2;i<=n;i++){
22            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])
23                ++tot;
24            x[sa[i]]=tot;
25        }
26        if(tot==n)break;else m=tot;
27    }
28 }
29 void calc_height(int n){
30    for(int i=1;i<=n;i++)rank[sa[i]]=i;
31    for(int i=1;i<=n;i++){
32        height[rank[i]]=max(0,height[rank[i-1]]-1);
33        if(rank[i]==1)continue;
34        int j=sa[rank[i]-1];
35        while(max(i,j)+height[rank[i]]<=n&&a[i+height[rank[i]]]==a[j+height[rank[i]]])
36            ++height[rank[i]];
37    }
38 }
```

最小表示法

```
1  int solve(char *text, int length) {//0-base ，多解答案为起点最小
2      int i = 0, j = 1, delta = 0;
3      while (i < length && j < length && delta < length) {
4          char tokeni = text[(i + delta) % length];
5          char tokenj = text[(j + delta) % length];
6          if (tokeni == tokenj) {
7              delta++;
8          } else {
9              if (tokeni > tokenj) {
10                 i += delta + 1;
11             } else {
12                 j += delta + 1;
13             }
14             if (i == j) {
15                 j++;
16             }
17             delta = 0;
18         }
19     }
20     return std::min(i, j);
21 }
```

Chapter 6

计算几何

点类

```cpp
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);
}
```

```
32 P operator/(P a,double b){
33     return P(a.x/b,a.y/b);
34 }
35 double operator^(P a,P b){
36     return a.x*b.x + a.y*b.y;
37 }
38 double operator*(P a,P b){
39     return a.x*b.y - a.y*b.x;
40 }
41 double det(P a,P b,P c){
42     return (b-a)*(c-a);
43 }
44 double dis(P a,P b){
45     return (b-a).len();
46 }
47 double Area(vector<P>poly){
48     double ans=0;
49     for(int i=1;i<poly.size();i++)
50         ans+=(poly[i]-poly[0])*(poly[(i+1)%poly.size()]-poly[0]);
51     return fabs(ans)/2;
52 }
53 struct L{
54     P a,b;
55     L(){}
56     L(P a,P b):a(a),b(b){}
57     P v(){return b-a;}
58 };
59 bool onLine(P p,L l){
60     return sgn((l.a-p)*(l.b-p))==0;
61 }
62 bool onSeg(P p,L s){
63     return onLine(p,s) and sgn((s.b-s.a)^(p-s.a))>=0 and sgn((s.a-s.b)^(p-s.b))>=0;
64 }
65 bool parallel(L l1,L l2){
66     return sgn(l1.v()*l2.v())==0;
67 }
68 P intersect(L l1,L l2){
69     double s1=det(l1.a,l1.b,l2.a);
70     double s2=det(l1.a,l1.b,l2.b);
71     return (l2.a*s2-l2.b*s1)/(s2-s1);
72 }
73 P project(P p,L l){
74     return l.a+l.v()*((p-l.a)^l.v())/l.v().len2();
75 }
76 double dis(P p,L l){
77     return fabs((p-l.a)*l.v())/l.v().len();
```

```
78  }
79  int dir(P p,L l){
80      int t=sgn((p-l.b)*(l.b-l.a));
81      if(t<0)return -1;
82      if(t>0)return 1;
83      return 0;
84  }
85  bool segIntersect(L l1,L l2){//strictly
86      if(dir(l2.a,l1)*dir(l2.b,l1)<0&&dir(l1.a,l2)*dir(l1.b,l2)<0)
87          return true;
88      return false;
89  }
90  bool in_tri(P pt,P *p){
91      if((p[1]-p[0])*(p[2]-p[0])<0)
92          reverse(p,p+3);
93      for(int i=0;i<3;i++){
94          if(!onLeft(pt,L(p[i],p[(i+1)%3])))
95              return false;
96      }
97      return true;
98  }
```

圆基础

```
1   struct C{
2       P o;
3       double r;
4       C(){}
5       C(P _o,double _r):o(_o),r(_r){}
6   };
7   // 求圆与直线的交点
8   //turn90() P(-y,x)
9   double fix(double x){return x>=0?x:0;}
10  bool intersect(C a, L l, P &p1, P &p2) {
11      double x = ((l.a - a.o)^ (l.b - l.a)),
12          y = (l.b - l.a).len2(),
13          d = x * x - y * ((l.a - a.o).len2() - a.r * a.r);
14      if (sgn(d) < 0) return false;
15      d = max(d, 0.0);
16      P p = l.a - ((l.b - l.a) * (x / y)), delta = (l.b - l.a) * (sqrt(d) / y);
17      p1 = p + delta, p2 = p - delta;
18      return true;
19  }
20  // 求圆与圆的交点，注意调用前要先判定重圆
21  bool intersect(C a, C b, P &p1, P &p2) {
```

```
22      double s1 = (a.o - b.o).len();
23      if (sgn(s1 - a.r - b.r) > 0 || sgn(s1 - fabs(a.r - b.r)) < 0) return false;
24      double s2 = (a.r * a.r - b.r * b.r) / s1;
25      double aa = (s1 + s2) * 0.5, bb = (s1 - s2) * 0.5;
26      P o = (b.o - a.o) * (aa / (aa + bb)) + a.o;
27      P delta = (b.o - a.o).norm().turn90() * sqrt(fix(a.r * a.r - aa * aa));
28      p1 = o + delta, p2 = o - delta;
29      return true;
30  }
31  // 求点到圆的切点，按关于点的顺时针方向返回两个点
32  bool tang(const C &c, const P &p0, P &p1, P &p2) {
33      double x = (p0 - c.o).len2(), d = x - c.r * c.r;
34      if (d < eps) return false; // 点在圆上认为没有切点
35      P p = (p0 - c.o) * (c.r * c.r / x);
36      P delta = ((p0 - c.o) * (-c.r * sqrt(d) / x)).turn90();
37      p1 = c.o + p + delta;
38      p2 = c.o + p - delta;
39      return true;
40  }
41  // 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
42  vector<L> extan(const C &c1, const C &c2) {
43      vector<L> ret;
44      if (sgn(c1.r - c2.r) == 0) {
45          P dir = c2.o - c1.o;
46          dir = (dir * (c1.r / dir.len())).turn90();
47          ret.push_back(L(c1.o + dir, c2.o + dir));
48          ret.push_back(L(c1.o - dir, c2.o - dir));
49      } else {
50          P p = (c1.o * -c2.r + c2.o * c1.r) / (c1.r - c2.r);
51          P p1, p2, q1, q2;
52          if (tang(c1, p, p1, p2) && tang(c2, p, q1, q2)) {
53              if (c1.r < c2.r) swap(p1, p2), swap(q1, q2);
54              ret.push_back(L(p1, q1));
55              ret.push_back(L(p2, q2));
56          }
57      }
58      return ret;
59  }
60  // 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
61  vector<L> intan(const C &c1, const C &c2) {
62      vector<L> ret;
63      P p = (c1.o * c2.r + c2.o * c1.r) / (c1.r + c2.r);
64      P p1, p2, q1, q2;
65      if (tang(c1, p, p1, p2) && tang(c2, p, q1, q2)) { // 两圆相切认为没有切线
66          ret.push_back(L(p1, q1));
67          ret.push_back(L(p2, q2));
```

```
68        }
69        return ret;
70  }
```

点在多边形内

```
1   bool InPoly(P p,vector<P>poly){
2       int cnt=0;
3       for(int i=0;i<poly.size();i++){
4           P a=poly[i],b=poly[(i+1)%poly.size()];
5           if(OnLine(p,L(a,b)))
6               return false;
7           int x=sgn(det(a,p,b));
8           int y=sgn(a.y-p.y);
9           int z=sgn(b.y-p.y);
10          cnt+=(x>0&&y<=0&&z>0);
11          cnt-=(x<0&&z<=0&&y>0);
12      }
13      return cnt;
14  }
```

二维最小覆盖圆

```
1   struct line{
2       point p,v;
3   };
4   point Rev(point v){return point(-v.y,v.x);}
5   point operator*(line A,line B){
6       point u=B.p-A.p;
7       double t=(B.v*u)/(B.v*A.v);
8       return A.p+A.v*t;
9   }
10  point get(point a,point b){
11      return (a+b)/2;
12  }
13  point get(point a,point b,point c){
14      if(a==b)return get(a,c);
15      if(a==c)return get(a,b);
16      if(b==c)return get(a,b);
17      line ABO=(line){(a+b)/2,Rev(a-b)};
18      line BCO=(line){(c+b)/2,Rev(b-c)};
19      return ABO*BCO;
20  }
21  int main(){
```

```
22    scanf("%d",&n);
23    for(int i=1;i<=n;i++)scanf("%lf%lf",&p[i].x,&p[i].y);
24    random_shuffle(p+1,p+1+n);
25    O=p[1];r=0;
26    for(int i=2;i<=n;i++){
27        if(dis(p[i],O)<r+1e-6)continue;
28        O=get(p[1],p[i]);r=dis(O,p[i]);
29        for(int j=1;j<i;j++){
30            if(dis(p[j],O)<r+1e-6)continue;
31            O=get(p[i],p[j]);r=dis(O,p[i]);
32            for(int k=1;k<j;k++){
33                if(dis(p[k],O)<r+1e-6)continue;
34                O=get(p[i],p[j],p[k]);r=dis(O,p[i]);
35            }
36        }
37    }printf("%.2lf %.2lf %.2lf\n",O.x,O.y,r);
38    return 0;
39 }s
```

半平面交

```
1  struct P{
2      int quad() const { return sgn(y) == 1 || (sgn(y) == 0 && sgn(x) >= 0);}
3  };
4  struct L{
5      bool onLeft(const P &p) const { return sgn((b - a)*( p - a)) > 0; }
6      L push() const{ // push out eps
7          const double eps = 1e-10;
8          P delta = (b - a).turn90().norm() * eps;
9          return L(a - delta, b - delta);
10     }
11 };
12 bool sameDir(const L &l0, const L &l1) {
13     return parallel(l0, l1) && sgn((l0.b - l0.a)^(l1.b - l1.a)) == 1;
14 }
15 bool operator < (const P &a, const P &b) {
16     if (a.quad() != b.quad())
17         return a.quad() < b.quad();
18     else
19         return sgn((a*b)) > 0;
20 }
21 bool operator < (const L &l0, const L &l1) {
22     if (sameDir(l0, l1))
23         return l1.onLeft(l0.a);
24     else
```

```
25        return (l0.b - l0.a) < (l1.b - l1.a);
26 }
27 bool check(const L &u, const L &v, const L &w) {
28     return w.onLeft(intersect(u, v));
29 }
30 vector<P> intersection(vector<L> &l) {
31     sort(l.begin(), l.end());
32     deque<L> q;
33     for (int i = 0; i < (int)l.size(); ++i) {
34         if (i && sameDir(l[i], l[i - 1])) {
35             continue;
36         }
37         while (q.size() > 1
38                 && !check(q[q.size() - 2], q[q.size() - 1], l[i]))
39                 q.pop_back();
40         while (q.size() > 1
41                 && !check(q[1], q[0], l[i]))
42                 q.pop_front();
43         q.push_back(l[i]);
44     }
45     while (q.size() > 2
46             && !check(q[q.size() - 2], q[q.size() - 1], q[0]))
47             q.pop_back();
48     while (q.size() > 2
49             && !check(q[1], q[0], q[q.size() - 1]))
50             q.pop_front();
51     vector<P> ret;
52     for (int i = 0; i < (int)q.size(); ++i)
53     ret.push_back(intersect(q[i], q[(i + 1) % q.size()]));
54     return ret;
55 }
```

求凸包

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

```
12      for(int i=(int)p.size()-1;i>=0;i--){
13          while(S.size()>=2
14                  && sgn(det(S[S.size()-2],S.back(),p[i]))<=0)
15                  S.pop_back();
16          S.push_back(p[i]);
17      }//up
18      for(int i=1;i+1<S.size();i++)
19          ans.push_back(S[i]);
20      return ans;
21 }
```

凸包游戏

```
1  /*
2      给定凸包，$\log n$ 内完成各种询问，具体操作有 :
3      1. 判定一个点是否在凸包内
4      2. 询问凸包外的点到凸包的两个切点
5      3. 询问一个向量关于凸包的切点
6      4. 询问一条直线和凸包的交点
7      INF 为坐标范围，需要定义点类大于号
8      改成实数只需修改 sign 函数，以及把 long long 改为 double 即可
9      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
10 */
11 const int INF = 1000000000;
12 struct Convex
13 {
14     int n;
15     vector<Point> a, upper, lower;
16     Convex(vector<Point> _a) : a(_a) {
17         n = a.size();
18         int ptr = 0;
19         for(int i = 1; i < n; ++ i) if (a[ptr] < a[i]) ptr = i;
20         for(int i = 0; i <= ptr; ++ i) lower.push_back(a[i]);
21         for(int i = ptr; i < n; ++ i) upper.push_back(a[i]);
22         upper.push_back(a[0]);
23     }
24     int sign(long long x) { return x < 0 ? -1 : x > 0; }
25     pair<long long, int> get_tangent(vector<Point> &convex, Point vec) {
26         int l = 0, r = (int)convex.size() - 2;
27         for( ; l + 1 < r; ) {
28             int mid = (l + r) / 2;
29             if (sign((convex[mid + 1] - convex[mid]).det(vec)) > 0) r = mid;
30             else l = mid;
31         }
32         return max(make_pair(vec.det(convex[r]), r)
```

```
33              , make_pair(vec.det(convex[0]), 0));
34      }
35      void update_tangent(const Point &p, int id, int &i0, int &i1) {
36          if ((a[i0] - p).det(a[id] - p) > 0) i0 = id;
37          if ((a[i1] - p).det(a[id] - p) < 0) i1 = id;
38      }
39      void binary_search(int l, int r, Point p, int &i0, int &i1) {
40          if (l == r) return;
41          update_tangent(p, l % n, i0, i1);
42          int sl = sign((a[l % n] - p).det(a[(l + 1) % n] - p));
43          for( ; l + 1 < r; ) {
44              int mid = (l + r) / 2;
45              int smid = sign((a[mid % n] - p).det(a[(mid + 1) % n] - p));
46              if (smid == sl) l = mid;
47              else r = mid;
48          }
49          update_tangent(p, r % n, i0, i1);
50      }
51      int binary_search(Point u, Point v, int l, int r) {
52          int sl = sign((v - u).det(a[l % n] - u));
53          for( ; l + 1 < r; ) {
54              int mid = (l + r) / 2;
55              int smid = sign((v - u).det(a[mid % n] - u));
56              if (smid == sl) l = mid;
57              else r = mid;
58          }
59          return l % n;
60      }
61      // 判定点是否在凸包内，在边界返回 true
62      bool contain(Point p) {
63          if (p.x < lower[0].x || p.x > lower.back().x) return false;
64          int id = lower_bound(lower.begin(), lower.end()
65              , Point(p.x, -INF)) - lower.begin();
66          if (lower[id].x == p.x) {
67              if (lower[id].y > p.y) return false;
68          } else if ((lower[id - 1] - p).det(lower[id] - p) < 0) return false;
69          id = lower_bound(upper.begin(), upper.end(), Point(p.x, INF)
70              , greater<Point>()) - upper.begin();
71          if (upper[id].x == p.x) {
72              if (upper[id].y < p.y) return false;
73          } else if ((upper[id - 1] - p).det(upper[id] - p) < 0) return false;
74          return true;
75      }
76      // 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
77      // 共线的多个切点返回任意一个，否则返回 false
78      bool get_tangent(Point p, int &i0, int &i1) {
```

```
79          if (contain(p)) return false;
80          i0 = i1 = 0;
81          int id = lower_bound(lower.begin(), lower.end(), p) - lower.begin();
82          binary_search(0, id, p, i0, i1);
83          binary_search(id, (int)lower.size(), p, i0, i1);
84          id = lower_bound(upper.begin(), upper.end(), p
85              , greater<Point>()) - upper.begin();
86          binary_search((int)lower.size() - 1, (int)lower.size() - 1 + id, p, i0, i1);
87          binary_search((int)lower.size() - 1 + id
88              , (int)lower.size() - 1 + (int)upper.size(), p, i0, i1);
89          return true;
90      }
91      // 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
92      int get_tangent(Point vec) {
93          pair<long long, int> ret = get_tangent(upper, vec);
94          ret.second = (ret.second + (int)lower.size() - 1) % n;
95          ret = max(ret, get_tangent(lower, vec));
96          return ret.second;
97      }
98      // 求凸包和直线 u,v 的交点，如果无严格相交返回 false.
99      //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
100     bool get_intersection(Point u, Point v, int &i0, int &i1) {
101         int p0 = get_tangent(u - v), p1 = get_tangent(v - u);
102         if (sign((v - u).det(a[p0] - u)) * sign((v - u).det(a[p1] - u)) < 0) {
103             if (p0 > p1) swap(p0, p1);
104             i0 = binary_search(u, v, p0, p1);
105             i1 = binary_search(u, v, p1, p0 + n);
106             return true;
107         } else {
108             return false;
109         }
110     }
111 };
```

平面最近点

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

```
10      d=min(solve(mid+1,r),d);
11      vector<P>tmp;
12      for(int i=l;i<=r;i++)
13          if(sqr(p[mid].x-p[i].x)<=d)
14              tmp.push_back(p[i]);
15      sort(tmp.begin(),tmp.end(),byY);
16      for(int i=0;i<tmp.size();i++)
17          for(int j=i+1;j<tmp.size()&&j-i<10;j++)
18              d=min(d,dis2(tmp[i],tmp[j]));
19      return d;
20 }
```

Chapter 7

技巧

无敌的读入优化

```cpp
// 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

```
1  template <typename T>
2  __inline void clear(T& container) {
3      container.clear(); // 或者删除了一堆元素
4      T(container).swap(container);
5  }
```

梅森旋转算法

```
1  template <typename T>
2  __inline void clear(T& container) {
3      container.clear(); // 或者删除了一堆元素
4      T(container).swap(container);
5  }
```

蔡勒公式

```
1  int solve(int year, int month, int day) {
2      int answer;
3      if (month == 1 || month == 2) {
4          month += 12;
5          year--;
6      }
7      if ((year < 1752) || (year == 1752 && month < 9) ||
8          (year == 1752 && month == 9 && day < 3)) {
9          answer = (day + 2 * month + 3 * (month + 1) / 5 + year + year / 4 + 5) % 7;
10     } else {
11         answer = (day + 2 * month + 3 * (month + 1) / 5 + year + year / 4
12                 - year / 100 + year / 400) % 7;
13     }
14     return answer;
15 }
```

开栈

```
1  register char *_sp __asm__("rsp");
2  int main() {
3      const int size = 400 << 20;//400MB
4      static char *sys, *mine(new char[size] + size - 4096);
5      sys = _sp; _sp = mine; _main(); _sp = sys;
6  }
```

Size 为 k 的子集

```
1  void solve(int n, int k) {
2      for (int comb = (1 << k) - 1; comb < (1 << n); ) {
3          // ...
4          int x = comb & -comb, y = comb + x;
5          comb = (((comb & ~y) / x) >> 1) | y;
6      }
7  }
```

长方体表面两点最短距离

```
1  int r;
2  void turn(int i, int j, int x, int y, int z,int x0, int y0, int L, int W, int H) {
3      if (z==0) { int R = x*x+y*y; if (R<r) r=R;
4      } else {
5          if(i>=0 && i< 2) turn(i+1, j, x0+L+z, y, x0+L-x, x0+L, y0, H, W, L);
6          if(j>=0 && j< 2) turn(i, j+1, x, y0+W+z, y0+W-y, x0, y0+W, L, H, W);
7          if(i<=0 && i>-2) turn(i-1, j, x0-z, y, x-x0, x0-H, y0, H, W, L);
8          if(j<=0 && j>-2) turn(i, j-1, x, y0-z, y-y0, x0, y0-H, L, H, W);
9      }
10 }
11 int main(){
12     int L, H, W, x1, y1, z1, x2, y2, z2;
13     cin >> L >> W >> H >> x1 >> y1 >> z1 >> x2 >> y2 >> z2;
14     if (z1!=0 && z1!=H) if (y1==0 || y1==W)
15         swap(y1,z1), std::swap(y2,z2), std::swap(W,H);
16     else swap(x1,z1), std::swap(x2,z2), std::swap(L,H);
17     if (z1==H) z1=0, z2=H-z2;
18     r=0x3fffffff;
19     turn(0,0,x2-x1,y2-y1,z2,-x1,-y1,L,W,H);
20     cout<<r<<endl;
21 }
```

经纬度求球面最短距离

```
1  double sphereDis(double lon1, double lat1, double lon2, double lat2, double R) {
2      return R * acos(cos(lat1) * cos(lat2) * cos(lon1 - lon2) + sin(lat1) * sin(lat2));
3  }
```

32-bit/64-bit 随机素数

32-bit	64-bit
73550053	12492928468556855773
148898719	17017504344199805569
189560747	36054998784244114901
459874703	56483166733887803781
1202316001	61253425708143357977
1431183547	62151553087758851301
1438011109	62946067780406234510
1538762023	63473305504460205470
1557944263	74296329243037252070
1981315913	85247200794803898490

NTT 素数及其原根

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

Formulas

Arithmetic Function

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

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

$J_k(n)$ 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Binomial Coefficients

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

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

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

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

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

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

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

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

$$\begin{bmatrix} n+1 \\ k \end{bmatrix} = n \begin{bmatrix} n \\ k \end{bmatrix} + \begin{bmatrix} n \\ k-1 \end{bmatrix}, \quad \begin{bmatrix} n+1 \\ 2 \end{bmatrix} = n! H_n$$

$$x^{\underline{n}} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} (-1)^{n-k} x^k, \quad x^{\overline{n}} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} x^k$$

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

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

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

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

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.

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

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 $\text{tv}(G) = \text{tw}(G)$ 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

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

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

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

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

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

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

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

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

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

Integrals of Rational Functions

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

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

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

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

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

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

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

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

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

Integrals with Roots

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

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

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

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

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

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

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

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

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

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

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

$$\int \frac{x}{\sqrt{x^2\pm a^2}}dx = \sqrt{x^2\pm a^2} \tag{21}$$

$$\int \frac{x}{\sqrt{a^2-x^2}}dx = -\sqrt{a^2-x^2} \tag{22}$$

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

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

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

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

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

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

Integrals with Logarithms

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

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

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

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

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

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

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

Integrals with Exponentials

$$\int x^n e^{ax}\,dx = \frac{x^n e^{ax}}{a} - \frac{n}{a}\int x^{n-1}e^{ax}\,dx \tag{36}$$

$$\int xe^{-ax^2}\,dx = -\frac{1}{2a}e^{-ax^2} \tag{37}$$

Integrals with Trigonometric Functions

$$\int \sin^3 ax\,dx = -\frac{3\cos ax}{4a} + \frac{\cos 3ax}{12a} \tag{38}$$

$$\int \cos^2 ax\,dx = \frac{x}{2} + \frac{\sin 2ax}{4a} \tag{39}$$

$$\int \cos^3 ax\,dx = \frac{3\sin ax}{4a} + \frac{\sin 3ax}{12a} \tag{40}$$

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

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

$$\int \sin^2 x\cos x\,dx = \frac{1}{...}\sin^3 x \tag{43}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Products of Trigonometric Functions and Monomials

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

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

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

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

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

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

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

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

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

```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
45     public int nextInt() {
46         return Integer.parseInt(next());
47     }
48
49     public long nextLong() {
50         return Long.parseLong(next());
51     }
52 }
```

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 $-2^{\text{Integer.MAX\_VALUE}}$ (exclusive) to $+2^{\text{Integer.MAX\_VALUE}}$ (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 $-2^{\text{Integer.MAX\_VALUE}}$ (exclusive) to $+2^{\text{Integer.MAX\_VALUE}}$ (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).
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

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.

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