# Algorithm Template

linxi

2020 年 10 月 4 日

## 目录

# 1 数论

## 1.1 素数筛

### 1.1.1 线性筛

```
const int maxn = 1e7 + 5;
const int maxp = 1e6 + 5;
// 1e7 以内的素数一共有 664579 （不到 7e5 ）个
int p[maxp], tot;
bool np[maxn];
void getPrime(int n) {
    tot = 0;
    for(int i = 0; i <= n; ++i) np[i] = false;
    for(int i = 2; i <= n; ++i) {
        if(!np[i]) p[tot++] = i;
        for(int j = 0; j < tot; ++j) {
            int x = i*p[j];
            if(x > n) break;
            np[x] = true;
            if(i%p[j] == 0) break;
        }
    }
}
```

### 1.1.2 小素数 x 在 log x 时间内分解

需对值域内所有的小素数线性预处理

```
const int maxn = 1e7 + 5;
const int maxp = 1e6 + 5;
int p[maxp], mf[maxn], tot;
bool np[maxn];
void getPrime(int n) {
    tot = 0;
    for(int i = 0; i <= n; ++i) np[i] = false;
    for(int i = 2; i <= n; ++i) {
        if(!np[i]) {
            p[tot++] = i;
            mf[i] = i;
        }
```

```
13          for(int j = 0; j < tot; ++j) {
14              int x = i*p[j];
15              if(x > n) break;
16              np[x] = true; mf[x] = p[j];
17              if(i%p[j] == 0) break;
18          }
19      }
20  }
21  int fac[35], num;
22  void getFactor(int x) {
23      num = 0;
24      while(x > 1) {
25          fac[num++] = mf[x];
26          x /= mf[x];
27      }
28  }
```

### 1.1.3 区间筛 ($O(nloglogn)$)

```
1   const int maxn = 1e6 + 5;
2   bool isP[maxn], isPS[maxn];
3   ll p[maxn];
4   int tot;
5   // O(nloglogn) 筛出在区间 [a, b) 中的所有素数
6   void SegmentSieve(ll a, ll b){
7       tot = 0;
8       for(ll i = 0; i*i < b; ++i) isPS[i] = true;
9       for(int i = 0; i < b - a; ++i) isP[i] = true;
10      for(ll i = 2; i*i < b; ++i){
11          if(!isPS[i]) continue;
12          for(ll j = 2*i; j*j < b; j += i) isPS[j] = false;
13          for(ll j = max(2LL, (a + i - 1)/i)*i; j < b; j += i) isP[j - a] =
                false;
14      }
15      for(ll i = 0; i < b - a; ++i){
16          if(!isP[i]) continue;
17          ll num = a + i;
18          if(num > 1) p[tot++] = a + i;
19      }
```

```
}
```

### 1.1.4 Meisell-Lehmer ($O(n^{\frac{2}{3}})$ 求 $[1,n]$ 素数的个数)

```
namespace Solver {
    const int N = 5e6 + 2;
    bool np[N];
    int prime[N], pi[N];
    int getprime() {
        int cnt = 0;
        np[0] = np[1] = 1;
        pi[0] = pi[1] = 0;
        for(int i = 2; i < N; ++i) {
            if(!np[i]) prime[++cnt] = i;
            pi[i] = cnt;
            for(int j = 1; j <= cnt && i * prime[j] < N; ++j) {
                np[i * prime[j]] = 1;
                if(i % prime[j] == 0)   break;
            }
        }
        return cnt;
    }
    const int M = 7;
    const int PM = 2 * 3 * 5 * 7 * 11 * 13 * 17;
    int phi[PM + 1][M + 1], sz[M + 1];
    void init() {
        getprime();
        sz[0] = 1;
        for(int i = 0; i <= PM; ++i)  phi[i][0] = i;
        for(int i = 1; i <= M; ++i) {
            sz[i] = prime[i] * sz[i - 1];
            for(int j = 1; j <= PM; ++j) phi[j][i] = phi[j][i - 1] - phi[j
                / prime[i]][i - 1];
        }
    }
    int sqrt2(ll x) {
        ll r = (ll)sqrt(x - 0.1);
        while(r * r <= x)   ++r;
        return int(r - 1);
```

```
35      }
36      int sqrt3(ll x) {
37          ll r = (ll)cbrt(x - 0.1);
38          while(r * r * r <= x)   ++r;
39          return int(r - 1);
40      }
41      ll getphi(ll x, int s) {
42          if(s == 0)  return x;
43          if(s <= M)  return phi[x % sz[s]][s] + (x / sz[s]) * phi[sz[s]][s
                ];
44          if(x <= prime[s]*prime[s])   return pi[x] - s + 1;
45          if(x <= prime[s]*prime[s]*prime[s] && x < N) {
46              int s2x = pi[sqrt2(x)];
47              ll ans = pi[x] - (s2x + s - 2) * (s2x - s + 1) / 2;
48              for(int i = s + 1; i <= s2x; ++i) ans += pi[x / prime[i]];
49              return ans;
50          }
51          return getphi(x, s - 1) - getphi(x / prime[s], s - 1);
52      }
53      ll getpi(ll x) {
54          if(x < N) return pi[x];
55          ll ans = getphi(x, pi[sqrt3(x)]) + pi[sqrt3(x)] - 1;
56          for(int i = pi[sqrt3(x)] + 1, ed = pi[sqrt2(x)]; i <= ed; ++i) ans
                -= getpi(x / prime[i]) - i + 1;
57          return ans;
58      }
59      ll lehmer_pi(ll x) { // get ans
60          if(x < N)   return pi[x];
61          int a = (int)lehmer_pi(sqrt2(sqrt2(x)));
62          int b = (int)lehmer_pi(sqrt2(x));
63          int c = (int)lehmer_pi(sqrt3(x));
64          ll sum = getphi(x, a) +(ll)(b + a - 2) * (b - a + 1) / 2;
65          for (int i = a + 1; i <= b; i++) {
66              ll w = x / prime[i];
67              sum -= lehmer_pi(w);
68              if (i > c) continue;
69              ll lim = lehmer_pi(sqrt2(w));
70              for (int j = i; j <= lim; j++) sum -= lehmer_pi(w / prime[j])
                    - (j - 1);
```

```
        }
        return sum;
    }
}
```

## 1.2 归并排序求逆序对 $O(nlog(n))$

```
// 下标, 例如数组int a[5], 全部排序的调用为 mgst(0, 4)
// a1为原数组, a2为临时数组
int a1[maxn], a2[maxn];
ll mg(int lf, int mid, int rt){
    int i = lf, j = mid + 1, k = lf;
    ll ret = 0;
    while(i <= mid && j <= rt){
        if(a1[i] <= a1[j]) {
            a2[k++] = a1[i++];
        } else{
            a2[k++] = a1[j++];
            ret += (ll)j - k;
        }
    }
    while(i <= mid) a2[k++] = a1[i++];
    while(j <= rt) a2[k++] = a1[j++];
    for(int i = lf; i <= rt; ++i) a1[i] = a2[i];
    return ret;
}

ll mgst(int lf, int rt){
    ll ret = 0;
    if(lf < rt){
        int mid = (lf + rt)/2;
        ret += mgst(lf, mid);
        ret += mgst(mid + 1, rt);
        ret += mg(lf, mid, rt);
    }
    return ret;
}
```

## 1.3 GCD 及其应用

### 1.3.1 拓展 GCD

```
//返回 d = gcd(a, b); 和对应于等式 ax + by = d 中的 x, y
ll ExGcd(ll a, ll b, ll &x, ll &y) {
    if(b == 0) {
        if(a == 0) return -1;
        x = 1; y = 0;
        return a;
    }
    ll d = ExGcd(b, a%b, y, x);
    y -= a/b*x;
    return d;
}
```

### 1.3.2 EXGCD 求逆元

```
// ax = 1(% mod)
ll mod_reverse(ll a, ll mod){ // 不要求mod为素数
    ll x, y;
    ll d = extend_gcd(a, mod, x, y);
    if(d == 1) return (x%mod + mod)%mod;
    return -1LL;
}
```

### 1.3.3 浮点数 GCD

```
int sgn(double x){
    if(fabs(x) < eps) return 0;
    if(x > 0) return 1;
    return -1;
}
double fgcd(double a, double b){
    if(sgn(a) == 0) return b;
    if(sgn(b) == 0) return a;
    return fgcd(b, fmod(a, b));
}
```

## 1.4 逆元

### 1.4.1 取模意义下快速幂 $O(log(n))$

```
ll pow_m(ll a, ll b, ll mod) { //取模意义下的 a 的 b 次方
    ll r = 1, bs = a;
    while(b) {
        if(b & 1)
            r = r*bs%mod;
        bs = bs*bs%mod;
        b >>= 1;
        b %= mod;
    }
    return r;
}
```

### 1.4.2 费马小定理

当模数为素数时，在模 $mod$ 意义下，$inv(a) = a^{-1} = a^{mod-2}$

### 1.4.3 线性递推阶乘逆元

```
const int maxn = 1e6 + 5;
const ll mod = 1e9 + 7;
ll fac[maxn], inv_fac[maxn];
ll pow_m(ll a, ll b, ll mod) { //取模意义下的 a 的 b 次方
    ll r = 1, bs = a;
    while(b) {
        if(b & 1)
            r = r*bs%mod;
        bs = bs*bs%mod;
        b >>= 1;
        b %= mod;
    }
    return r;
}
void getFacInv(int n) { // n 为最大所需阶乘数
    fac[0] = 1;
    for(int i = 1; i <= n; ++i)
        fac[i] = (fac[i-1]*i)%mod;
    inv_fac[n] = pow_m(fac[n], mod - 2, mod);
```

```
    for(int i = n-1; i >= 0; --i)
        inv_fac[i] = (inv_fac[i+1]*(i+1))%mod;
    return ;
}
```

#### 1.4.4 线性递推逆元

```
int inv[maxn];
void GetInv(int p) {
    inv[1] = 1;
    for(int i = 2; i < p; ++i) {
        inv[i] = (ll)(p - (p/i))*inv[p%i]%p;
        printf("inv[%d] = %d\n", i, inv[i]);
    }
}
```

### 1.5 中国剩余定理 CRT

#### 1.5.1 CRT（要求模数数组两两互质）

```
const int maxn = 1e5 + 5;
int a[maxn]; // 余数数组
int m[maxn]; // 模数数组

// n为数组长度
ll CRT(int n) {
    ll M = 1;
    ll ans = 0;
    for(int i = 0; i < n; ++i)
        M *= m[i];
    for(int i = 0; i < n; ++i) {
        ll Mi = M/m[i];
        ans = (ans+(((Mi*inv(Mi, m[i]))%M)*a[i]%M))%M;
    }
    while(ans < 0) ans += M;
    return ans;
}
```

### 1.5.2 EXCRT（不要求模数数组两两互质）

```
/*
 *   hdu 1573
 *   求在小于等于 n 的正整数中有多少个 X 满足:
 *   X mod a[0] = b[0], X mod a[1] = b[1], ..... , X mod a[m - 1] = b[m - 1]
 *   (0 < n <= 1,000,000,000, 0 < m <= 10)
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 10 + 5;
ll a[maxn], b[maxn], n, m;
ll extgcd(ll a, ll b, ll &x, ll &y){
    ll d = a;
    if(b != 0){
        d = extgcd(b, a%b, y, x);
        y -= (a/b)*x;
    }
    else{
        x = 1;
        y = 0;
    }
    return d;
}
ll ExCRT(){
    ll a1 = a[0], b1 = b[0];
    for(int i = 1; i < m; i++){
        ll a2 = a[i], b2 = b[i];
        ll bb = b2 - b1, x, y;
        ll d = extgcd(a1, a2, x, y);
        if(bb%d) return 0;
        ll k = bb/d*x, t = a2/d;
        if(t < 0) t = -t;
        k = (k%t + t)%t;
        b1 = b1 + a1*k;
        a1 = a1/d*a2;
    }
    ll ret = b1;
```

```
38      if (ret == 0) ret = a1;
39      if(ret > n) return 0;
40      ret = (n - ret)/a1 + 1;
41      return ret;
42  }
43
44  int main(){
45      int t; scanf("%d", &t);
46      while(t--){
47          scanf("%lld %lld", &n, &m);
48          for(int i = 0; i < m; ++i) scanf("%lld", &a[i]);
49          for(int i = 0; i < m; ++i) scanf("%lld", &b[i]);
50          printf("%lld\n", ExCRT());
51      }
52      return 0;
53  }
```

## 1.6 欧拉函数

### 1.6.1 线性筛

```
1   const int maxn = 1e6 + 5;
2   int phi[maxn], p[maxn], tot;
3   bool np[maxn];
4   void get_phi(int n) {
5       tot = 0;
6       phi[1] = 1;
7       for (int i = 2; i <= n; ++i) {
8           if (!np[i]) {
9               p[tot++] = i;
10              phi[i] = i - 1;
11          }
12          for (int j = 0; j < tot; ++j) {
13              int x = i*p[j];
14              if (x > n) break;
15              np[x] = true;
16              if (i%p[j] == 0) {
17                  phi[x] = phi[i]*p[j];
18                  break;
19              } else {
```

```
                phi[x] = phi[i]*(p[j] - 1);
            }
        }
    }
}
```

### 1.6.2 $O(\sqrt{x})$ 求单个欧拉函数值

```
ll get_phi(ll x) {
    ll ret = x, sx = sqrt(x) + 1;
    for (ll j = 2; j <= sx; ++j) {
        if (x%j == 0) {
            ret = ret/j*(j - 1);
            while (x%j == 0) x /= j;
        }
    }
    if (x > 1) ret = ret/x*(x - 1LL);
    return ret;
}
```

## 1.7 莫比乌斯函数

如果一个数包含平方因子，那么 miu(n) = 0。

例如：miu(4), miu(12), miu(18) = 0。

如果一个数不包含平方因子，并且有 k 个不同的质因子，那么 $miu(n) = (-1)^k$。

例如：miu(2), miu(3), miu(30) = -1,miu(1), miu(6), miu(10) = 1。

### 1.7.1 线性筛

```
const int maxn = 1e7 + 5;
bool isNotPrime[maxn];
vector<int> prime;
int miu[maxn];
void getMiu(int n) {
    isNotPrime[1] = true;
    miu[1] = 1;
    for(int i = 2; i <= n; i++) {
        if(!isNotPrime[i]) {
            prime.push_back(i);
            miu[i] = -1;
```

```
12          }
13          for(int j = 0; j < prime.size() && i*prime[j] <= n; ++j){
14              isNotPrime[i*prime[j]] = true;
15              if(i%prime[j] == 0) {
16                  miu[i*prime[j]] = 0;
17                  break;
18              }
19              miu[i*prime[j]] = -miu[i];
20          }
21      }
22  }
```

### 1.7.2 $O(\sqrt{x})$ 求单个莫比乌斯函数的值

```
1   int getMiu(ll n){
2       int v = 1;
3       for(ll i = 2; i*i <= n; ++i){
4           if(n%i == 0){
5               v = -v;
6               n /= i;
7               if(n%i == 0) return 0;
8           }
9       }
10      if(n != 1) v = -v;
11      return v;
12  }
```

### 1.7.3 数论分块

对于给出的 n 个询问，每次求有多少个数对 $(x,y)$，满足 $a \le x \le b$，$c \le y \le d$，且 $gcd(x,y) = k$，$gcd(x,y)$ 函数为 x 和 y 的最大公约数。

```
1   int tot;
2   int miu[maxn], p[maxn], sum[maxn];
3   bool notP[maxn];
4   void getMiu() {
5       miu[1] = 1; tot = 0;
6       sum[0] = 0; sum[1] = 1;
7       for(int i = 2; i < maxn; ++i) {
8           if(!notP[i]) {
```

```
9               p[tot++] = i;
10              miu[i] = -1;
11          }
12          for(int j = 0; j < tot; ++j) {
13              int num = i*p[j];
14              if(num >= maxn) break;
15              notP[num] = true;
16              if(i%p[j] == 0) {
17                  miu[num] = 0;
18                  break;
19              }
20              miu[num] = -miu[i];
21          }
22          sum[i] = sum[i - 1] + miu[i];
23      }
24  }
25  int solve(int a, int b, int d) {
26      if(a > b) swap(a, b);
27      a /= d; b /= d;
28      int ret = 0;
29      for(int lf = 1, rt; lf <= a; lf = rt + 1) {
30          int qa = a/lf, qb = b/lf;
31          int ra = a/qa, rb = b/qb;
32          rt = min(ra, rb);
33          ret += (sum[rt] - sum[lf - 1])*qa*qb;
34      }
35      return ret;
36  }
37  int main() {
38      getMiu();
39      int t; scanf("%d", &t);
40      while(t--) {
41          int a, b, c, d, k; scanf("%d%d%d%d%d", &a, &b, &c, &d, &k);
42          int ans = solve(b, d, k) - solve(a - 1, d, k) - solve(b, c - 1, k)
                + solve(a - 1, c - 1, k);
43          printf("%d\n", ans);
44      }
45      return 0;
46  }
```

## 1.8 离散对数 EXBSGS 算法

已知 $a, b, Pa, b, P$, a 与 P 不一定互质

求解同余方程 $ax\ b(modP)$

```
class Hash {
    static const int HashMod = 3000017;
    // 99991, 3000017, 7679977, 19260817
    int tp, st[HashMod + 5];
    bool vis[HashMod + 5];
    ll h[HashMod + 5];
    ll val[HashMod + 5];
    int locate(const ll& x) const {
        int p = x%HashMod;
        while(vis[p] && h[p] != x) {
            if(++p == HashMod) p = 0;
        }
        return p;
    }
public:
    Hash() {
        tp = 0;
        memset(vis, false, sizeof(false));
    }
    void ist(const ll& x, const ll& y) {
        const int p = locate(x);
//          assert(!vis[p] || h[p] == x);
        h[p] = x; val[p] = y;
        vis[p] = true; st[++tp] = p;
    }
    bool get(const ll& x, ll& y) const {
        const int p = locate(x);
        if(vis[p] && h[p] == x) {
            y = val[p];
            return true;
        }
        return false;
    }
    void clr() {
        while(tp) vis[st[tp--]] = false;
    }
```

```
};
Hash H;
ll ExGcd(ll a, ll b, ll &x, ll &y) {
    if(b == 0) {
        if(a == 0) return -1;
        x = 1; y = 0;
        return a;
    }
    ll d = ExGcd(b, a%b, y, x);
    y -= a/b*x;
    return d;
}
ll ExBSGS(ll A, ll B, ll C) {
    H.clr();
    ll ret = 1;
    for(ll i = 0; i <= 50; ++i) {
        if(ret == B) return i;
        ret = ret*A%C;
    }
    ll d, x, y, ans = 1, cnt = 0;
    while((d = ExGcd(A, C, x, y)) != 1) {
        if(B%d) return -1;
        C /= d; B /= d;
        ans = ans*(A/d)%C;
        ++cnt;
    }
    ll m = (ll)ceil(sqrt(C*1.0)), t = 1;
    for(ll i = 0; i < m; ++i) {
        H.ist(t, i);
        t = t*A%C;
    }
    for(ll i = 0, j; i < m; ++i) {
        ExGcd(ans, C, x, y);
        ll val = x*B%C;
        val = (val%C + C)%C;
        if(H.get(val, j))  return m*i + j + cnt;
        ans = ans*t%C;
    }
    return -1;
```

```
}
```

## 1.9 欧拉降幂公式

欧拉定理：若正整数 $a,n$ 互质，则 $a^{Euler(n)}\ 1(modn)$

推论：若正整数 $a,n$ 互质，则对于任意正整数 $b$, 有 $a^b\ a^{b\%Euler(n)}(modn)$

对于 $p$ 是素数，$Euler(p)=p-1$ 且只有 $p$ 的倍数与 $p$ 不互质

```
#define Mod(a, b) (a < b ? a : a%b + b)
unordered_map<ll, ll> p;
ll Pow(ll x, ll y, ll mod){
    ll ret = 1;
    while(y){
        if(y & 1) ret = Mod(ret*x, mod);
        x = Mod(x*x, mod);
        y >>= 1;
    }
    return ret;
}
ll phi(ll k){
    ll s = k, x = k;
    if(p[k] != 0) return p[k];
    for(ll i = 2; i*i <= k; ++i){
        if(k%i == 0) s = s/i*(i - 1);
        while(k%i == 0) k /= i;
    }
    if(k > 1) s = s/k*(k - 1);
    p[x] = s;
    return s;
}
ll a[maxn];
// return a[lf] ^ a[lf + 1] ^ ... ^ a[rt]
// 注意返回值需要再 % mod , 因为 Mod 被重载为欧拉降幂的形式
ll solve(int lf, int rt, ll mod){
    if(lf == rt || mod == 1) return Mod(a[lf], mod);
    return Pow(a[lf], solve(lf + 1, rt, phi(mod)), mod);
}
```

## 1.10 因子相关

### 1.10.1 因子个数

$[1, 5 \times 10^6]$ 中，4324320 的因子数最多，有 384 个因子

### 1.10.2 因子和线性筛

```
const int maxn = 4e6 + 5;
bool np[maxn];
int p[maxn], tot;
ll sf[maxn], sp[maxn]; // sf: sum of factor sp: sum of prime
void init() {
    tot = 0; sf[1] = 1;
    for(int i = 2; i < maxn; ++i) {
        if(!np[i]) {
            p[tot++] = i;
            sf[i] = i + 1;
            sp[i] = i + 1;
        }
        for(int j = 0; j < tot && i*p[j] < maxn; ++j) {
            np[i*p[j]] = true;
            if(i%p[j] == 0) {
                sp[i*p[j]] = sp[i]*p[j] + 1;
                sf[i*p[j]] = sf[i]/sp[i]*sp[i*p[j]];
                break;
            }
            sf[i*p[j]] = sf[i]*sf[p[j]];
            sp[i*p[j]] = 1LL + p[j];
        }
    }
}
```

## 1.11 威尔逊定理（利用阶乘判定素数的充要条件）

在初等数论中，威尔逊定理给出了判定一个自然数是否为素数的充分必要条件。
即当且仅当 $p$ 为素数时：

$$(p-1)! \ -1(modp)$$

## 1.12 斯特林近似（n 的阶乘的长度）

斯特林公式 $n! = sqrt(2 * PI * n) * (n/e)^n$

```
const double PI = acos(-1.0);
ll solve(int n){
    double nn = n;
    ll ret = (ll)((0.5*log(2.0*PI*nn) + nn*log(nn) - nn)/log(10.0));
    ++ret;
    return ret;
}
```

## 1.13 MIN25 筛

```
namespace MIN25 {
    ll w[N], s1[N], s2[N];
    int Sqr, plen, m, p[N];

    int getid(ll x, ll n) {
        return x <= Sqr ? x : m - n / x + 1;
    }

    int S(ll n) {
        if (n & 1)
            return (n + 1) / 2 % MOD * (n % MOD) % MOD;
        return n / 2 % MOD * ((n + 1) % MOD) % MOD;
    }

    int f(int n, int c) {
        return (n ^ c) % MOD;
    }

    void pre(ll n) {
        Sqr = sqrt(n), m = 0;
        for (ll l = 1, r; l <= n; l = r + 1) {
            r = n / (n / l);
            w[++m] = r;
            s1[m] = r, s2[m] = S(r);
        }
        p[0] = 1;
        for(int i = 2; i <= Sqr; i++) {
```

```
28              if (s1[i] != s1[i - 1]) {
29                  p[++plen] = i;
30                  ll lim = (ll)i * i;
31                  for (int j = m; lim <= w[j]; j--) {
32                      int id1 = getid(w[j] / i, n), id2 = p[plen - 1];
33                      s1[j] -= (s1[id1] - s1[id2]);
34                      s2[j] -= (ll)i * (s2[id1] - s2[id2]);
35                  }
36                  s2[i] %= MOD;
37              }
38          }
39          for(int i = 1; i <= m; i++) {
40              s1[i] = (s2[i] - s1[i] + (i > 1) * 2) % MOD;
41          }
42      }
43
44      ll sieve(ll n, int y) {
45          if (n <= 1 || p[y] > n)
46              return 0;
47          pre(n);
48          ll ans = s1[getid(n, n)] - s1[p[y - 1]];
49          for (int i = y; i <= plen && (ll)p[i] * p[i] <= n; i++) {
50              for (ll P = p[i], j = 1; P * p[i] <= n; P *= p[i], j++)
51                  ans += sieve(n / P, i + 1) * f(p[i], j) + f(p[i], j + 1);
52          }
53          return ans % MOD;
54      }
55  }
```

# 2 线性代数

## 2.1 矩阵快速幂

### 2.1.1 模板

```
1  #define mod(x) ((x) % MOD)
2  const ll MOD = 1e9 + 7;
3  const int maxn = 1e2 + 5;
4  int n;// 矩阵大小
5  struct mat {
```

```
6       ll m[maxn][maxn];
7       mat operator *(const mat& a) const { //  矩阵乘法
8           mat ret;
9           for(int i = 0; i < n; ++i) {
10              for(int j = 0; j < n; ++j) {
11                  ll x = 0;
12                  for(int k = 0; k < n; ++k) {
13                      x += mod(m[i][k]*a.m[k][j]);
14                      x = mod(x);
15                  }
16                  ret.m[i][j] = mod(x);
17              }
18          }
19          return ret;
20      }
21  } unit; //  单元矩阵
22  void init_unit() {
23      for(int i = 0; i < n; i++) unit.m[i][i] = 1;
24      return ;
25  }
26  // 调用 powMat() 前应首先初始化单位矩阵 init_unit()
27  mat powMat(mat a, ll x) {
28      mat ret = unit;
29      while(x){
30          if(x&1) ret = ret*a;
31          x >>= 1;
32          a = a*a;
33      }
34      return ret;
35  }
```

### 2.1.2 斐波那契数列第 N 项

斐波那契数列矩阵形式

$$M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$F_n = M^n[1][0]$$

通过矩阵快速幂求解

### 2.1.3 一般线性递推式

$$f_i = a_1 f_{i-1} + a_2 f_{i-2} + \cdots + a_k f_{i-k}$$

$$A = \begin{bmatrix} a_1 & a_2 & a_3 & \cdots & a_k \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}$$

### 2.1.4 矩阵幂的和

求解 $S = A + A^2 + A^3 + ..... + A^k, (1 <= n <= 30, 1 <= k <= 10^9, 1 <= M <= 10^4)$

令

$S_k = I + \cdots + A^{k-1}$

则有

$$\begin{bmatrix} A_k \\ S_k \end{bmatrix} = \begin{bmatrix} A & 0 \\ I & I \end{bmatrix}^k \begin{bmatrix} I \\ 0 \end{bmatrix}$$

```
#include <iostream>
#include <cstdio>
using namespace std;
const int mod = 10;
int N;
struct Mat {
    int a[85][85];
    Mat() {
        for(int i = 0; i < N; ++i) {
            for(int j = 0; j < N; ++j) {
                a[i][j] = 0;
            }
        }
    }
    Mat operator *(const Mat& rhs) const {
        Mat ret;
        for(int i = 0; i < N; ++i) {
            for(int j = 0; j < N; ++j) {
                int& x = ret.a[i][j];
                x = 0;
                for(int k = 0; k < N; ++k) {
                    x = (x + a[i][k]*rhs.a[k][j])%mod;
                }
```

```
24              }
25          }
26          return ret;
27      }
28  };
29  void init_unit(Mat& unit) {
30      for(int i = 0; i < N; ++i) {
31          for(int j = 0; j < N; ++j) {
32              unit.a[i][j] = 0;
33          }
34          unit.a[i][i] = 1;
35      }
36  }
37  Mat pw(Mat a, int n, Mat& unit) {
38      Mat ret = unit;
39      while(n > 0) {
40          if(n & 1) ret = ret*a;
41          a = a*a;
42          n >>= 1;
43      }
44      return ret;
45  }
46  int main() {
47      int n, k;
48      while(scanf("%d%d", &n, &k) != EOF) {
49          if(n == 0) break;
50          N = 2*n;
51          Mat a, unit;
52          init_unit(unit);
53          for(int i = 0; i < n; ++i) {
54              for(int j = 0; j < n; ++j) {
55                  scanf("%d", &a.a[i][j]);
56                  a.a[i][j] %= mod;
57              }
58          }
59          for(int i = 0; i < n; ++i) {
60              a.a[i + n][i] = 1;
61              a.a[i + n][i + n] = 1;
62          }
```

```
63          Mat b = pw(a, k + 1, unit);
64          for(int i = 0; i < n; ++i) {
65              for(int j = 0; j < n; ++j) {
66                  int x = b.a[i + n][j];
67                  if(i == j) x = (x - 1 + mod)%mod;
68                  if(j > 0) printf(" ");
69                  printf("%d", x);
70              }
71              puts("");
72          }
73          puts("");
74      }
75      return 0;
76  }
```

## 2.2 BM 算法（线性递推式）

```
1  typedef long long ll;
2  #define rep(i,a,n) for (int i=a;i<n;i++)
3  #define per(i,a,n) for (int i=n-1;i>=a;i--)
4  #define pb push_back
5  #define mp make_pair
6  #define all(x) (x).begin(),(x).end()
7  #define fi first
8  #define se second
9  #define SZ(x) ((ll)(x).size())
10 typedef vector<ll> VI;
11 typedef pair<ll,ll> PII;
12 const ll mod = 1e9+7;
13 ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0); for(;b;b>>=1){if(b&1)
       res=res*a%mod;a=a*a%mod;}return res;}
14 // head
15
16 ll _,n;
17 namespace linear_seq {
18     const int N=10010;
19     ll res[N],base[N],_c[N],_md[N];
20
21     vector<ll> Md;
```

```
22      void mul(ll *a,ll *b,ll k) {
23          rep(i,0,k+k) _c[i]=0;
24          rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
25          for (ll i=k+k-1;i>=k;i--) if (_c[i])
26              rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[j]]-_c[i]*_md[Md[j]])
                    %mod;
27          rep(i,0,k) a[i]=_c[i];
28      }
29      ll solve(ll n,VI a,VI b) {
30  //          a 系数 b 初值 b[n+1]=a[0]*b[n]+...
31          for(int i = 0; i < a.size(); i++)
32              printf("sss  %d\n", a[i]);
33          ll ans=0,pnt=0;
34          ll k=SZ(a);
35          assert(SZ(a)==SZ(b));
36          rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
37          Md.clear();
38          rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
39          rep(i,0,k) res[i]=base[i]=0;
40          res[0]=1;
41          while ((1ll<<pnt)<=n) pnt++;
42          for (ll p=pnt;p>=0;p--) {
43              mul(res,res,k);
44              if ((n>>p)&1) {
45                  for (ll i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;
46                  rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%
                        mod;
47              }
48          }
49          rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
50          if (ans<0) ans+=mod;
51          return ans;
52      }
53      VI BM(VI s) {
54          VI C(1,1),B(1,1);
55          ll L=0,m=1,b=1;
56          rep(n,0,SZ(s)) {
57              ll d=0;
58              rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
```

```
59              if (d==0) ++m;
60              else if (2*L<=n) {
61                  VI T=C;
62                  ll c=mod-d*powmod(b,mod-2)%mod;
63                  while (SZ(C)<SZ(B)+m) C.pb(0);
64                  rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
65                  L=n+1-L; B=T; b=d; m=1;
66              } else {
67                  ll c=mod-d*powmod(b,mod-2)%mod;
68                  while (SZ(C)<SZ(B)+m) C.pb(0);
69                  rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
70                  ++m;
71              }
72          }
73          return C;
74      }
75      ll gao(VI a,ll n) {
76          VI c=BM(a);
77          c.erase(c.begin());
78          rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
79          return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
80      }
81  };
82
83  int main() {
84      while (~scanf("%d",&n)) {
85          vector<int>v;
86          v.push_back(1);
87          v.push_back(2);
88          v.push_back(4);
89          v.push_back(7);
90          v.push_back(13);
91          v.push_back(24);
92          //VI{1,2,4,7,13,24}
93          printf("%d\n", linear_seq::gao(v,n-1));
94      }
95  }
```

## 2.3 高斯消元

高斯消元基本思想是将方程式的系数和常数化为矩阵，通过将矩阵通过行变换成为阶梯状（上三角矩阵），然后从小往上逐一求解。

### 2.3.1 代数意义下

求解下列线性方程组：

a[0][0]*x0 + a[0][1]*x1 + ... + a[0][n - 1]*xn - 1 == a[0][n] ... ... a[n - 1][0]*x0 + a[n - 1][1]*x1 + ... + a[n - 1][n - 1]*xn - 1 == a[n - 1][n]

最终结果为：

x0 = a[0][n] ... ... xn - 1 = a[n - 1][n]

若无解，则返回 false

```
1  double a[maxn][maxn];
2  bool gauss(int n) {
3      for(int i = 0; i < n; ++i){
4          int r = i;
5          for(int j = i + 1; j < n; j++) if(fabs(a[j][i]) > fabs(a[r][i])) r
               = j;
6  /*
7  * 浮点高斯消元中找到最大绝对值的系数来消元本来是为了优化精度
8  * 如果在整数的模意义下，这种行为就没有意义了
9  * 此时我们只需要找到第i行及以下任意一个系数非零的元来消即可。
10 * for(int j = i ; j < n; j++) if(a[j][i]!=0) { r = j; break;}
11 */
12         if(r != i) for(int j = 0; j <= n; j++) swap(a[i][j], a[r][j]);
13
14         for(int j = n; j >= i; --j)
15             for(int k = i + 1; k < n; ++k)
16                 a[k][j] -= a[k][i]/a[i][i]*a[i][j];
17     }
18     for(int i = n - 1 ; i >= 0 ; --i){
19         for(int j = i + 1 ; j < n ; j++){
20             double val = a[i][j]*a[j][n];
21             a[i][n] -= val;
22         }
23         if(fabs(a[i][i]) < 1e-10){
24             if(fabs(a[i][n]) > 1e-10) return false;
25             continue;
26         }
27         a[i][n] = a[i][n]/a[i][i];
```

```
    }
    return true;
}
```

## 2.4 线性基（区间子集异或最值）

```
struct Linear_Basis {
    int tv, tb; // total variable, total base
    ll b[64], nb[64];
    void init() {
        tv = tb = 0;
        memset(b, 0, sizeof(b));
        memset(nb, 0, sizeof(nb));
    }
    bool ist(ll x){
        ++tv;
        for(int i = 62; i >= 0; --i) {
            if (x & (1LL << i)) {
                if(!b[i]) {
                    b[i] = x;
                    break;
                }
                x ^= b[i];
            }
        }
        if(x <= 0) return false;
        return true;
    }
    ll Max(ll x) {
        ll ret = x;
        for (int i = 62; i >= 0; --i) ret = max(ret, ret ^ b[i]);
        return ret;
    }
    ll Min(ll x) {
        ll ret = x;
        for (int i = 0; i <= 62; ++i)
            if (b[i]) ret ^= b[i];
        return ret;
    }
```

```
34      void rebuild() {
35          for (int i = 62; i >= 0; --i)
36              for (int j = i - 1; j >= 0; --j)
37                  if (b[i] & (1LL << j)) b[i] ^= b[j];
38          tb = 0;
39          for (int i = 0; i <= 62; ++i)
40              if (b[i]) nb[tb++] = b[i];
41      }
42      ll Kth_Max(ll k) {
43          if (tv != tb) --k;
44          if (k >= (1LL << tb)) return -1LL;
45          ll ret = 0;
46          for (int i = 62; i >= 0; --i)
47              if (k & (1LL << i)) ret ^= nb[i];
48          return ret;
49      }
50  } LB;
```

## 2.5 线性规划问题（单纯形法）

例题 bzoj 3112: [Zjoi2013] 防守战线

战线可以看作一个长度为 n 的序列，现在需要在这个序列上建塔来防守敌兵，在序列第 i 号位置上建一座塔有 Ci 的花费，且一个位置可以建任意多的塔费用累加计算。有 m 个区间 [L1, R1], [L2, R2], …, [Lm, Rm]，在第 i 个区间的范围内要建至少 Di 座塔。求最少花费。

对其对偶问题使用单纯形法

standard:

$maxz = c_0 * x_0 + ... + c_{n-1} * x_{n-1}$

$a_{00} * x_0 + a_{01} * x_1 ... + a_{0n-1} * x_{n-1} <= b_0$

...

$a_{m-10} * x_0 + a_{m1} * x_1 ... + a_{m-1n-1} * x_{n-1} <= b_{m-1}$

```
1   const double M = 1e100;
2   const double thres = 1e50;
3   const double eps = 1e-6;
4   int sgn(double x) {
5       if(fabs(x) < eps) return 0;
6       if(x > 0) return 1;
7       return -1;
8   }
9   namespace simplex {
```

```
10      const int maxr = 2e3 + 5;
11      const int maxc = 2e4 + 5;
12      int row, column;
13      double a[maxr][maxc], b[maxr], c[maxc], s[maxc];
14      int rid[maxr], cid[maxc];
15      void output() {
16          puts("");
17          printf("row: %d column: %d\n", row, column);
18          puts("++++++++++++++++++++++++++++++++++++++++");
19          printf("max z = ");
20          for(int i = 0; i < column; ++i) {
21              if(i > 0) printf(" + ");
22              printf("%.2f*x_%d", c[i], i);
23          }
24          puts("");
25          puts("subject to: ");
26          for(int i = 0; i < row; ++i) {
27              for(int j = 0; j < column; ++j) {
28                  if(j > 0) printf(" + ");
29                  printf("%.2f*x_%d", a[i][j], j);
30              }
31              printf(" = %.2f\n", b[i]);
32          }
33          puts("++++++++++++++++++++++++++++++++++++++++");
34          printf("row id:");
35          for(int i = 0; i < row; ++i) {
36              printf(" %d", rid[i]);
37          }
38          puts("");
39          printf("column id:");
40          for(int i = 0; i < column; ++i) {
41              printf(" %d", cid[i]);
42          }
43          puts("");
44          puts("++++++++++++++++++++++++++++++++++++++++");
45          double z = 0.0;
46          printf("z = ");
47          for(int i = 0; i < row; ++i) {
48              if(i > 0) printf("+ ");
```

```
49              if(rid[i] < column) {
50                  printf("%.2f*%.2f ", b[i], c[rid[i]]);
51                  z += b[i]*c[rid[i]];
52              } else {
53                  printf("%.2f*%.2f ", b[i], 0.0);
54              }
55          }
56          printf("= %.2f\n", z);
57          puts("++++++++++++++++++++++++++++++++++++++++");
58          puts("");
59      }
60      void init(int m, int n) {
61          row = m; column = n + 1;
62          c[n] = -M;
63          for(int i = 0; i < row; ++i) {
64              a[i][n] = -1.0;
65              rid[i] = column + i;
66          }
67          for(int i = 0; i < column; ++i) {
68              cid[i] = i;
69              s[i] = c[i];
70          }
71          // output();
72      }
73      bool getRC(int &rv, int &cv) {
74          rv = -1; cv = -1;
75          double sigma = -M;
76          for(int i = 0; i < column; ++i) {
77              if(cv == -1 || s[i] > sigma) {
78                  sigma = s[i];
79                  cv = i;
80              }
81          }
82          // printf("cv: %d sigma: %.2f\n", cv, sigma);
83          if(sgn(sigma) <= 0) return 1;
84          double theta = M;
85          for(int i = 0; i < row; ++i) {
86              if(sgn(a[i][cv]) <= 0) continue;
87              double ntheta = b[i]/a[i][cv];
```

```
88              if(rv == -1 || sgn(ntheta - theta) < 0) {
89                  theta = ntheta;
90                  rv = i;
91              }
92          }
93          if(theta > thres) return 2;
94          return 0;
95      }
96      void pivot(const int &rv, const int &cv) {
97          assert(rv != -1 && cv != -1);
98          b[rv] /= a[rv][cv];
99          for(int i = 0; i < column; ++i) {
100             if(i == cv) continue;
101             a[rv][i] /= a[rv][cv];
102         }
103         a[rv][cv] = 1.0/a[rv][cv];
104         for(int i = 0; i < row; ++i) {
105             if(i == rv || sgn(a[i][cv]) == 0) continue;
106             b[i] -= a[i][cv]*b[rv];
107             for(int j = 0; j < column; ++j) {
108                 if(j == cv) continue;
109                 a[i][j] -= a[i][cv]*a[rv][j];
110             }
111             a[i][cv] = -a[i][cv];
112         }
113         for(int i = 0; i < column; ++i) {
114             if(i == cv) continue;
115             s[i] -= s[cv]*a[rv][i];
116         }
117         s[cv] = -s[cv]*a[rv][cv];
118         swap(rid[rv], cid[cv]);
119     }
120     int solve(int m, int n, double &z) {
121         init(m, n);
122         while(true) {
123             int rv, cv;
124             int sta = getRC(rv, cv);
125             if(sta == 2) return 2;
126             if(sta == 1) break;
```

```
127                 pivot(rv, cv);
128             }
129             z = 0.0;
130             for(int i = 0; i < row; ++i) {
131                 if(rid[i] >= column) continue;
132                 z += b[i]*c[rid[i]];
133             }
134             if(z < -thres) return 1;
135             return 0;
136         }
137     }
138
139     int main() {
140         int n, m; scanf("%d%d", &n, &m);
141         for(int i = 0; i < n; ++i) {
142             scanf("%lf", &simplex::b[i]);
143         }
144         for(int i = 0; i < m; ++i) {
145             int l, r; scanf("%d%d%lf", &l, &r, &simplex::c[i]);
146             --l, --r;
147             for(int j = 0; j < n; ++j) {
148                 if(j >= l && j <= r) {
149                     simplex::a[j][i] = 1.0;
150                 } else {
151                     simplex::a[j][i] = 0.0;
152                 }
153             }
154         }
155         double ans = 0.0;
156         int sta = simplex::solve(n, m, ans);
157         if(sta == 0) {
158             printf("%.0f\n", ans);
159             // printf("max z = %.2f\n", ans);
160         } else if(sta == 1) {
161             assert(false);
162             // printf("no feasible solution\n");
163         } else {
164             assert(false);
165             // printf("unbounded solution. (z -> inf)\n");
```

```
166     }
167     return 0;
168 }
169 /*
170 input:
171     5 3
172     1 5 6 3 4
173     2 3 1
174     1 5 4
175     3 5 2
176 output:
177     11
178 */
```

# 3 组合数学

## 3.1 组合数

### 3.1.1 模板

```
1  const int maxn = 1e6 + 5;
2  const ll mod = 1e9 + 7;
3  ll fac[maxn], invFac[maxn];
4  ll pw(ll a, ll b){
5      ll ret = 1;
6      while(b){
7          if(b&1) ret = (ret*a)%mod;
8          a = (a*a)%mod;
9          b >>= 1;
10     }
11     return ret;
12 }
13 void getFacInv(int n){
14     fac[0] = 1;
15     for(int i = 1; i <= n; ++i) fac[i] = (fac[i - 1]*i)%mod;
16     invFac[n] = pw(fac[n], mod - 2);
17     for(int i = n - 1; i >= 0; --i) invFac[i] = (invFac[i + 1]*(i + 1))%
           mod;
18 }
19 ll C(int a, int b){
```

```
20      if(b < 0 || b > a) return 0LL;
21      ll ret = (fac[a]*invFac[b])%mod;
22      ret = (ret*invFac[a - b])%mod;
23      return ret;
24  }
```

### 3.1.2 Lucas 定理（求解 n, m 大，小模数取模的组合数）

在取模意义下：

$C(n,m) = (C(n\%mod, m\%mod) * C(n/mod, m/mod))\%mod$

```
1   ll Lucas(ll a, ll b){
2       if(b == 0) return 1;
3       ll ret = (C(a%mod, b%mod, mod)*Lucas(a/mod, b/mod))%mod;
4       return ret;
5   }
```

## 3.2 康托展开与逆康托展开（全排列到自然数的双射）

```
1   int a[12], fac[12];
2   bool vis[12];
3   void getfac(int n){
4       fac[0] = 1;
5       for(int i = 1; i < n; ++i) fac[i] = fac[i - 1]*i;
6   }
7   int cantor(int n){
8       int ret = 0;
9       for(int i = 0; i < n; ++i){
10          int cnt = 0;
11          for(int j = i + 1; j < n; ++j) if(a[i] > a[j]) ++cnt;
12          ret += cnt*fac[n - 1 - i];
13      }
14      return ret;
15  }
16  void decantor(int x, int n){
17      for(int i = 1; i <= n; ++i) vis[i] = false;
18      int tot = 0;
19      for(int i = n; i >= 1; --i){
20          int q = x/fac[i - 1];
21          int cnt = 0;
```

```
22          for(int j = 1; j <= n; ++j){
23              if(vis[j]) continue;
24              if(++cnt > q){
25                  a[tot++] = j;
26                  vis[j] = true;
27                  break;
28              }
29          }
30          x = x%fac[i - 1];
31      }
32  }
```

## 3.3 快速傅里叶变换 FFT （加速加法操作卷积运算 O(nlogn)）

```
1   const int maxn = 2e5 + 5;
2   const int fMaxn = maxn << 2; // fMaxn >= 4*maxn (1025 need 4096)
3   namespace FFT {
4       // typedef long double ld;
5       const double pi = acos(-1.0);
6       struct Complex {
7           double r, i;
8           Complex(double r = 0, double i = 0):r(r), i(i) {};
9           Complex operator + (const Complex & rhs) {
10              return Complex(r + rhs.r, i + rhs.i);
11          }
12          Complex operator - (const Complex & rhs) {
13              return Complex(r - rhs.r, i - rhs.i);
14          }
15          Complex operator * (const Complex &rhs) {
16              return Complex(r * rhs.r - i * rhs.i, i * rhs.r + r * rhs.i);
17          }
18      } va[fMaxn], vb[fMaxn], vc[fMaxn];
19      void change(Complex F[], int len) {
20          int j = len >> 1;
21          for(int i = 1; i < len - 1; ++i) {
22              if(i < j) {
23                  swap(F[i], F[j]);
24              }
25              int k = len >> 1;
```

```
26              while(j >= k) {
27                  j -= k;
28                  k >>= 1;
29              }
30              if(j < k) {
31                  j += k;
32              }
33          }
34      }
35      void fft(Complex F[], int len, int t) {
36          change(F, len);
37          for(int h = 2; h <= len; h <<= 1) {
38              Complex wn(cos(-t*2.0*pi/h), sin(-t*2.0*pi/h));
39              for(int j = 0; j < len; j += h) {
40                  Complex E(1, 0);
41                  for(int k = j; k < j + h/2; ++k) {
42                      Complex u = F[k];
43                      Complex v = E*F[k + h/2];
44                      F[k] = u + v;
45                      F[k + h/2] = u - v;
46                      E = E*wn;
47                  }
48              }
49          }
50          if(t == -1) {
51              for(int i = 0; i < len; i++) {
52                  F[i].r /= len;
53              }
54          }
55      }
56      template<class T>
57      int init(T a[], T b[], int la, int lb) {
58          int lc = 1, mi = 2*max(la, lb);
59          while(lc < mi) {
60              lc <<= 1;
61          }
62          for(int i = 0; i < la; ++i) {
63              va[i] = {(double)a[i], 0.0};
64          }
```

```
65          for(int i = la; i < lc; ++i) {
66              va[i] = {0.0, 0.0};
67          }
68          for(int i = 0; i < lb; ++i) {
69              vb[i] = {(double)b[i], 0.0};
70          }
71          for(int i = lb; i < lc; ++i) {
72              vb[i] = {0.0, 0.0};
73          }
74          return lc;
75      }
76      template<class T>
77      int solve(T a[], T b[], T c[], int la, int lb) { // id: [0, la), [0,
            lb), [0, lc)
78          int lc = init(a, b, la, lb);
79          fft(va, lc, 1);
80          fft(vb, lc, 1);
81          for(int i = 0; i < lc; ++i) {
82              vc[i] = va[i]*vb[i];
83          }
84          fft(vc, lc, -1);
85          for(int i = 0; i < lc; ++i) {
86              c[i] = vc[i].r; // double
87              // c[i] = (int)round(vc[i].r); // int
88              // c[i] = (ll)round(vc[i].r); // ll
89          }
90          return lc;
91      }
92  }
```

### 3.4 n 个 m 边形划分平面

用 n 个 m 边方形最多可以把平面分成 S(n, m) = 2 + (n - 1)*n*m 个区域

### 3.5 高维前缀和（对子集（超集）求和）

```
1   void GetSum(int n, int mx) {
2       // sum[S] = \Sigma sum[T], T \in S （T 是 S 的子集）
3       // 对子集求和
4       // for(int i = 0; i < n; ++i) {
```

```
    //      for(int j = 0; j <= mx; ++j) {
    //          if((j & (1 << i)) != 0) sum[j] += sum[j ^ (1 << i)];
    //      }
    // }

    // sum[S] = \Sigma sum[T], S \in T （T 是 S 的超集）
    // 对超集求和
    for(int i = 0; i < n; ++i) {
        for(int j = 0; j <= mx; ++j) {
            if((j & (1 << i)) != 0) sum[j ^ (1 << i)] += sum[j];
        }
    }
}
// GetSum(int(floor(log2(1000000)) + 1), 1000000);
// GetSum(n, (1 << n) - 1);
```

## 3.6 快速沃尔什变换 FWT （加速位操作卷积操作 O(nlogn)）

$$H[x] = \sum_{iOPj=x} F[i] * G[j]$$

的位操作卷积（OP 代表一种位运算，比如：与、或、异或）

```
const int mod = 1e9 + 7;
// inv2 = pw(2, p - 2, p);
const int inv2 = 5e8 + 4;
// ty == 1: FWT, ty == -1: IFWT
void FWT(int a[], int n, int ty){
    for(int d = 1; d < n; d <<= 1)
        for(int m = d << 1, i = 0; i < n; i += m)
            for(int j = 0; j < d; ++j){
                int x = a[i + j], y = a[i + j + d];
                if(ty == 1) {
                    a[i + j] = (x + y)%mod;
                    a[i + j + d] = (x - y + mod)%mod;
                    //xor: a[i + j] = x + y; a[i + j + d] = x - y;
                    //and: a[i + j] = x + y;
                    //or: a[i + j + d] = x + y;
                } else {
                    a[i + j] = 1LL*(x + y)*inv2%mod;
```

```
                    a[i + j + d] = (1LL*(x - y)*inv2%mod + mod)%mod;
                    //xor: a[i + j] = (x + y)/2; a[i + j + d] = (x - y)/2;
                    //and: a[i + j] = x - y;
                    //or: a[i + j + d] = y - x;
                }
            }
}
void solve(int a[], int b[], int n){
    FWT(a, n, 1);
    FWT(b, n, 1);
    for(int i = 0; i < n; ++i) {
        a[i] = 1LL*a[i]*b[i]%mod;
    }
    FWT(a, n, -1);
}
```

## 3.7 整数划分

### 3.7.1 划分为不同整数

将 $N(1 <= N <= 50000)$ 分为若干个不同整数的和，有多少种不同的划分方式，例如：n = 6，6 1, 5 2, 4 1, 2, 3，共 4 种。由于数据较大，输出 Mod $10^9 + 7$ 的结果即可。

$F[i][j]$ 表示将数 $i$ 分为 $j$ 份的方案数总和。

于是便有了状态转移方程：

$$f[i][j] := f[i - j][j] + f[i - j][j - 1]$$

下面来具体解释一下这个方程:

1、$f[i - j][j]$ 表示的是将 i 分为不包含 1(min >= 2) 的方案总个数，例如，6( = 9 - 3) 分成 3 份可以分为 1, 2, 3，则 9 可以分为 1 + 1, 2 + 1, 3 + 1 -> 2, 3, 4 仅 1 种

2、$f[i - j][j - 1]$ 表示的是将 i 分为包含 1 (min = 1) 的方案总个数，例如，6( = 9 - 3) 分成 2( = 3 - 1) 可以分为 0, 1, 50, 2, 4

则 9 可以分为 0 + 1, 1 + 1, 5 + 1 0 + 1, 2 + 1, 4 + 1 -> 1, 2, 61, 3, 5（共 2 种）

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int mod = 1e9 + 7;
const int maxn = 5e4 + 5;
int dp[maxn][405];
int main() {
```

```
    int n; scanf("%d", &n);
    int m = sqrt(2*n) + 1;
    dp[0][0] = 1;
    for(int i = 1; i <= n; ++i) {
        for(int j = 1; j <= m; ++j) {
            if(j > i) break;
            dp[i][j] = (dp[i - j][j] + dp[i - j][j - 1])%mod;
        }
    }
    int ans = 0;
    for(int i = 1; i <= m; ++i) ans = (ans + dp[n][i])%mod;
    printf("%d\n", ans);
    return 0;
}
```

### 3.7.2 可存在相同整数

将 $N(1 <= N <= 50000)$ 分为若干个整数的和，有多少种不同的划分方式，例如：n = 4，4 1,3 2,2 1,1,2 1,1,1,1，共 5 种。由于数据较大，输出 Mod $10^9 + 7$ 的结果即可。

欧拉的五边形定理：

$P(n)$ 表示 $n$ 的划分种数

$$P(n) = \sum_{k \geq 1} (P(n - \frac{k(3k-1)}{2}) + P(n - \frac{k(3k+1)}{2}))$$

n < 0 时，P(n) = 0, n = 0 时, P(n) = 1

```
const int mod = 1e9 + 7;
const int maxn = 5e4 + 5;
int dp[maxn];
void solve() {
    int n; scanf("%d", &n);
    dp[0] = 1;
    for(int i = 1; i <= n; ++i) {
        dp[i] = 0;
        for(int j = 1; ; ++j) {
            int k1 = j*(3*j - 1)/2;
            int k2 = j*(3*j + 1)/2;
            if(k1 > i && k2 > i) break;
            int f;
            if(j%2 == 1) f = 1;
```

```
15              else f = -1;
16              if(k1 <= i) dp[i] += f*dp[i - k1];
17              dp[i] %= mod;
18              if(k2 <= i) dp[i] += f*dp[i - k2];
19              dp[i] %= mod;
20              if(dp[i] < 0) dp[i] += mod;
21          }
22      }
23      printf("%d\n", dp[n]);
24  }
```

# 4 几何基础

## 4.1 点和向量

```
1   struct Point{
2       double x, y;
3       Point(double x = 0, double y = 0):x(x),y(y){}
4   };
5   typedef Point Vector;
6   Vector operator + (Vector A, Vector B){
7       return Vector(A.x+B.x, A.y+B.y);
8   }
9   Vector operator - (Point A, Point B){
10      return Vector(A.x-B.x, A.y-B.y);
11  }
12  Vector operator * (Vector A, double p){
13      return Vector(A.x*p, A.y*p);
14  }
15  Vector operator / (Vector A, double p){
16      return Vector(A.x/p, A.y/p);
17  }
18  bool operator < (const Point& a, const Point& b){
19      if(a.x == b.x)
20          return a.y < b.y;
21      return a.x < b.x;
22  }
23  const double eps = 1e-6;
24  int sgn(double x){
```

```
    if(fabs(x) < eps) return 0;
    if(x < 0) return -1;
    return 1;
}
bool operator == (const Point& a, const Point& b){
    if(sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0) return true;
    return false;
}

double Dot(Vector A, Vector B){
    return A.x*B.x + A.y*B.y;
}
double Length(Vector A){
    return sqrt(Dot(A, A));
}
double Angle(Vector A, Vector B){
    return acos(Dot(A, B) / Length(A) / Length(B));
}
double Cross(Vector A, Vector B){
    return A.x*B.y-A.y*B.x;
}
double Area2(Point A, Point B, Point C){
    return Cross(B-A, C-A);
}
Vector Rotate(Vector A, double rad){//rad为弧度 且为逆时针旋转的角
    return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
}
Vector Normal(Vector A){//向量A左转90°的单位法向量
    double L = Length(A);
    return Vector(-A.y/L, A.x/L);
}
bool ToLeftTest(Point a, Point b, Point c){
    return Cross(b - a, c - b) > 0;
}
```

## 4.2 复数类实现平面向量

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

```
typedef complex<double> Point;
// 复数定义向量后，自动拥有构造函数、加减法和数量积
typedef Point Vector;
const double eps = 1e-9;
int sgn(double x){
    if(fabs(x) < eps)
        return 0;
    if(x < 0)
        return -1;
    return 1;
}
double Length(Vector A){
    return abs(A);
}
// conj(a + bi) 返回共轭复数 a - bi
double Dot(Vector A, Vector B){
    return real(conj(A)*B);
}
double Cross(Vector A, Vector B){
    return imag(conj(A)*B);
}
// exp(p) 返回以 e 为底复数的指数
Vector Rotate(Vector A, double rad){
    return A*exp(Point(0, rad));
}
```

## 4.3 点和直线

### 4.3.1 直线定义

```
struct Line{
    Point v, p;
    Line(Point v, Point p):v(v), p(p) {}
    Point point(double t){ // 返回点 P = v + (p - v)*t
        return v + (p - v)*t;
    }
};
```

### 4.3.2 计算两直线交点

```
// 调用前需保证 Cross(v, w) != 0
Point GetLineIntersection(Point P, Vector v, Point Q, Vector w){
    Vector u = P-Q;
    double t = Cross(w, u)/Cross(v, w);
    return P + v*t;
}
```

#### 4.3.3 点 P 到直线 AB 距离公式

```
// 不取绝对值, 得到的是有向距离
double DistanceToLine(Point P, Point A, Point B) {
    Vector v1 = B - A, v2 = P - A;
    return fabs(Cross(v1, v2)/Length(v1));
}
```

#### 4.3.4 点 P 到线段 AB 的距离公式

```
double DistanceToSegment(Point P, Point A, Point B) {
    if(A == B) return Length(P - A);
    Vector v1 = B - A, v2 = P - A, v3 = P - B;
    if(sgn(Dot(v1, v2)) < 0) return Length(v2);
    if(sgn(Dot(v1, v3)) > 0) return Length(v3);
    return DistanceToLine(P, A, B);
}
```

#### 4.3.5 点 P 在直线 AB 上的投影点

```
Point GetLineProjection(Point P, Point A, Point B){
    Vector v = B-A;
    return A + v*(Dot(v, P-A)/Dot(v, v));
}
```

#### 4.3.6 判断 p 点是否在线段 a1a2 上

```
bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2){
    double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1);
    double c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1, a2-b1);
    // if 判断控制是否允许线段在端点处相交, 根据需要添加
```

```
    if(!sgn(c1) || !sgn(c2) || !sgn(c3) || !sgn(c4)){
        bool f1 = OnSegment(b1, a1, a2);
        bool f2 = OnSegment(b2, a1, a2);
        bool f3 = OnSegment(a1, b1, b2);
        bool f4 = OnSegment(a2, b1, b2);
        bool f = (f1 | f2 | f3 | f4);
        return f;
    }
    return (sgn(c1)*sgn(c2) < 0 && sgn(c3)*sgn(c4) < 0);
}
```

## 4.4 求三角形的外心、垂心、重心、费马点

```
struct Point {
    double x, y;
};
struct Line{
    Point a, b;
};
Point Intersection(Line u, Line v){
    Point ret = u.a;
    double t1 = (u.a.x - v.a.x)*(v.a.y - v.b.y) - (u.a.y - v.a.y)*(v.a.x -
        v.b.x);
    double t2 = (u.a.x - u.b.x)*(v.a.y - v.b.y) - (u.a.y - u.b.y)*(v.a.x -
        v.b.x);
    double t = t1/t2;
    ret.x += (u.b.x - u.a.x)*t;
    ret.y += (u.b.y - u.a.y)*t;
    return ret;
}
// 外心
Point Circumcenter(Point a, Point b, Point c){
    Line u, v;
    u.a.x = (a.x + b.x)/2;
    u.a.y = (a.y + b.y)/2;
    u.b.x = u.a.x - a.y + b.y;
    u.b.y = u.a.y + a.x - b.x;
    v.a.x = (a.x + c.x)/2;
    v.a.y = (a.y + c.y)/2;
```

```
25      v.b.x = v.a.x - a.y + c.y;
26      v.b.y = v.a.y + a.x - c.x;
27      return Intersection(u, v);
28  }
29  // 垂心
30  Point Perpencenter(Point a, Point b, Point c){
31      Line u, v;
32      u.a = c;
33      u.b.x = u.a.x - a.y + b.y;
34      u.b.y = u.a.y + a.x - b.x;
35      v.a = b;
36      v.b.x = v.a.x - a.y + c.y;
37      v.b.y = v.a.y + a.x - c.x;
38      return Intersection(u, v);
39  }
40  // 三角形重心
41  // 到三角形三顶点距离的平方和最小的点
42  // 三角形内到三边距离之积最大的点
43  Point barycenter(Point a, Point b, Point c){
44      Line u, v;
45      u.a.x = (a.x + b.x)/2;
46      u.a.y = (a.y + b.y)/2;
47      u.b = c;
48      v.a.x = (a.x + c.x)/2;
49      v.a.y = (a.y + c.y)/2;
50      v.b = b;
51      return Intersection(u, v);
52  }
53  inline double Dist(Point p1, Point p2){
54      return sqrt((p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y))
            ;
55  }
56  // 三角形费马点
57  // 到三角形三顶点距离之和最小的点
58  Point Ferment(Point a, Point b, Point c){
59      Point u, v;
60      double step = fabs(a.x) + fabs(a.y) + fabs(b.x) + fabs(b.y) + fabs(c.x
            ) + fabs(c.y);
61      u.x = (a.x + b.x + c.x)/3;
```

```
62      u.y = (a.y + b.y + c.y)/3;
63      while(step >1e-10){
64          for(int k = 0; k < 10; step /= 2, k++){
65              for(int i = -1; i <= 1; ++i){
66                  for(int j = -1; j <= 1; ++j){
67                      v.x = u.x + step*i;
68                      v.y = u.y + step*j;
69                      double t1 = Dist(u, a) + Dist(u, b) + Dist(u, c);
70                      double t2 = Dist(v, a) + Dist(v, b) + Dist(v, c);
71                      if (t1 > t2) u = v;
72                  }
73              }
74          }
75      }
76      return u;
77  }
```

## 4.5 多边形

### 4.5.1 多边形有向面积

```
1   // p 为端点集合, n 为端点个数
2   double PolygonArea(Point *p, int n){
3       double s = 0;
4       for(int i = 1; i < n-1; ++i)
5           s += Cross(p[i]-p[0], p[i+1]-p[0]);
6       return s;
7   }
```

### 4.5.2 判断点是否在不自交多边形（可以为凹多边形）内

```
1   // 判断点是否在多边形内, 若点在多边形内返回1, 在多边形外部返回0, 在多边形
        上返回-1
2   int isPointInPolygon(Point p, vector<Point> poly){
3       int wn = 0;
4       int n = poly.size();
5       for(int i = 0; i < n; ++i){
6           if(OnSegment(p, poly[i], poly[(i+1)%n])) return -1;
7           int k = sgn(Cross(poly[(i+1)%n] - poly[i], p - poly[i]));
```

```
8           int d1 = sgn(poly[i].y - p.y);
9           int d2 = sgn(poly[(i+1)%n].y - p.y);
10          if(k > 0 && d1 <= 0 && d2 > 0) wn++;
11          if(k < 0 && d2 <= 0 && d1 > 0) wn--;
12      }
13      if(wn != 0) return 1;
14      return 0;
15  }
```

### 4.5.3 计算不自交多边形与直线相交长度和

```
1   const int maxn = 1e3 + 5;
2   struct Point{
3       double x, y;
4       Point(double x = 0, double y = 0):x(x),y(y){}
5   } p[maxn];
6   typedef Point Vector;
7   Vector operator - (Point A, Point B){
8       return Vector(A.x-B.x, A.y-B.y);
9   }
10  const double eps = 1e-5;
11  int sgn(double x){
12      if(fabs(x) < eps)
13          return 0;
14      if(x < 0)
15          return -1;
16      return 1;
17  }
18  double Dot(Vector A, Vector B){
19      return A.x*B.x + A.y*B.y;
20  }
21  double Length(Vector A){
22      return sqrt(Dot(A, A));
23  }
24  double Cross(Vector A, Vector B){
25      return A.x*B.y-A.y*B.x;
26  }
27  struct node{
28      double len;
```

```
29      int cs;
30      node(double len = 0, int cs = 0):len(len), cs(cs){}
31      bool operator < (const node& nd) const{
32          return len < nd.len;
33      }
34  };
35  double cal(Point A, Point B, Point C, Point D){
36      double ret = Cross(C - A, D - C)/Cross(B - A, D - C);
37      return ret;
38  }
39  vector<node> v;
40  // n: size of polygon a, b: points on line
41  double solve(int n, Point A, Point B){
42      double len = Length(A - B);
43      v.clear();
44      for(int i = 0; i < n; ++i){
45          int a = sgn(Cross(p[i] - A, B - A));
46          int b = sgn(Cross(p[(i + 1)%n] - A, B - A));
47          if(a == b) continue;
48          v.push_back(node(len*cal(A, B, p[i], p[(i + 1)%n]), a - b));
49      }
50      int sz = (int)v.size();
51      sort(v.begin(), v.end());
52      int wn = 0;
53      double ret = 0;
54      for(int i = 0; i < sz - 1; ++i){
55          wn += v[i].cs;
56          if(wn) ret += v[i + 1].len - v[i].len;
57      }
58      return ret;
59  }
60  int main(){
61      int n, m; scanf("%d%d", &n, &m);
62      for(int i = 0; i < n; ++i) scanf("%lf%lf", &p[i].x, &p[i].y);
63      while(m--){
64          Point A, B;
65          scanf("%lf%lf%lf%lf", &A.x, &A.y, &B.x, &B.y);
66          double ans = solve(n, A, B);
67          printf("%.20f\n", ans);
```

```
68      }
69      return 0;
70  }
```

## 4.6 圆

### 4.6.1 常用定义

```
1  struct Circle{
2      Point c; double r;
3      Circle(Point c, double r):c(c), r(r) {}
4      Point point(double a){//通过圆心角求坐标
5          return Point(c.x + cos(a)*r, c.y + sin(a)*r);
6      }
7  };
```

### 4.6.2 求圆与直线交点

```
1  // 求圆与直线交点
2  int getLineCircleIntersection(Line L, Circle C, double& t1, double& t2,
       vector<Point>& sol){
3      double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;
4      double e = a*a + c*c, f = 2*(a*b + c*d), g = b*b + d*d - C.r*C.r;
5      double delta = f*f - 4*e*g; // 判别式
6      if(sgn(delta) < 0) // 相离
7          return 0;
8      if(sgn(delta) == 0){ // 相切
9          t1 = -f /(2*e);
10         t2 = -f /(2*e);
11         sol.push_back(L.point(t1));//sol存放交点本身
12         return 1;
13     }
14     // 相交
15     t1 = (-f - sqrt(delta))/(2*e);
16     sol.push_back(L.point(t1));
17     t2 = (-f + sqrt(delta))/(2*e);
18     sol.push_back(L.point(t2));
19     return 2;
20 }
```

### 4.6.3 求两圆相交面积

```
double AreaOfOverlap(Point c1, double r1, Point c2, double r2){
    double d = Length(c1 - c2);
    if(r1 + r2 < d + eps)
        return 0.0;
    if(d < fabs(r1 - r2) + eps){
        double r = min(r1, r2);
        return pi*r*r;
    }
    double x = (d*d + r1*r1 - r2*r2)/(2.0*d);
    double p = (r1 + r2 + d)/2.0;
    double t1 = acos(x/r1);
    double t2 = acos((d - x)/r2);
    double s1 = r1*r1*t1;
    double s2 = r2*r2*t2;
    double s3 = 2*sqrt(p*(p - r1)*(p - r2)*(p - d));
    return s1 + s2 - s3;
}
```

## 4.7 网格图

### 4.7.1 pick 定理

Pick 定理是指一个计算点阵中顶点在格点上的多边形面积公式该公式可以表示为

$$2S = 2a + b - 2$$

其中 a 表示多边形内部的点数，b 表示多边形边界上的点数，S 表示多边形的面积。
常用形式

$$S = a + 2b - 1$$

### 4.7.2 多边形内部和边上网格点数量

```
struct Point{
    int x, y;
};
// 多边形上的网格点个数
int Onedge(int n, Point* p){
int re  t = 0;
```

```
7       for(int i = 0; i < n; ++i)
8           ret += __gcd(abs(p[i].x - p[(i + 1)%n].x), abs(p[i].y - p[(i + 1)%
                n].y));
9       return ret;
10  }
11  // 多边形内的网格点个数
12  int Inside(int n, Point* p){
13      int ret = 0;
14      for(int i = 0; i < n; ++i)
15          ret += p[(i + 1)%n].y*(p[i].x - p[(i + 2)%n].x);
16      ret = (abs(ret) - Onedge(n, p))/2 + 1;
17      return ret;
18  }
```

# 5 计算几何

## 5.1 凸包 Andrew 算法

```
1   //计算凸包, 输入点数组为 p, 个数为 n, 输出点数组为 ch。函数返回凸包顶点数
2   struct Point {
3       double x, y;
4       Point(double x = 0, double y = 0) :x(x), y(y) {}
5   };
6   typedef Point Vector;
7   const double eps = 1e-6;
8   bool operator < (const Point& a, const Point& b) {
9       if(a.x == b.x)
10          return a.y < b.y;
11      return a.x < b.x;
12  }
13  int sgn(double x) {
14      if(fabs(x) < eps) return 0;
15      if(x < 0) return -1;
16      return 1;
17  }
18  double Cross(Vector A, Vector B) {
19      return A.x * B.y - A.y * B.x;
20  }
21  int ConvexHull(Point* p, int n, Point* ch) {
```

```
22      std::sort(p, p + n);
23      int m = 0;
24      for(int i = 0; i < n; ++i) {
25          while(m > 1 && Cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2]) < 0)
                --m;
26          ch[m++] = p[i];
27      }
28      int k = m;
29      for(int i = n - 2; i >= 0; --i) {
30          while(m > k && Cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2]) < 0)
                --m;
31          ch[m++] = p[i];
32      }
33      if(n > 1) --m;
34      return m;
35  }
```

## 5.2 旋转卡壳

### 5.2.1 求凸包直径

```
1   // 计算距离的平方
2   double Dist2(Point p1, Point p2) {
3       double ret = Dot(p1 - p2, p1 - p2);
4       return ret;
5   }
6   // 返回平面最大距离的平方
7   double RotatingCalipers(Point* ch, int m) {
8       if(m == 1) return 0.0;
9       if(m == 2) return Dist2(ch[0], ch[1]);
10      double ret = 0.0;
11      ch[m] = ch[0];
12      int j = 2;
13      for(int i = 0; i < m; ++i) {
14          while(Cross(ch[i + 1] - ch[i], ch[j] - ch[i]) < Cross(ch[i + 1] -
                ch[i], ch[j + 1] - ch[i]))
15              j = (j + 1)%m;
16          ret = max(ret, max(Dist2(ch[j], ch[i]), Dist2(ch[j], ch[i + 1])));
17      }
18      return ret;
```

```
}
```

### 5.2.2 求凸包的宽（凸包对踵点的最小值）

```
double dist(Point a, Point b){
    double ret = sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y));
    return ret;
}
double mxid[maxn];
double RotateCalipersWide(Point* p, int n){
    int j = 1;
    p[n] = p[0];
    for(int i = 0; i < n; ++i){
        while(fabs(Cross(p[i + 1] - p[i], p[j + 1] - p[i])) > fabs(Cross(p
            [i + 1] - p[i], p[j] - p[i]))) j = (j + 1)%n;
        mxid[i] = fabs(Cross(p[j] - p[i], p[i + 1] - p[i]))/dist(p[i], p[i
            + 1]);
    }
    double ret = 1e18;
    for(int i = 0; i < n; ++i) ret = min(ret, mxid[i]);
    return ret;
}
```

### 5.2.3 凸包上最大三角形面积

```
double RotatingCalipers(Point* p, int n){
    if(n < 3) return 0.0;
    double ret = 0;
    for(int i = 0; i < n; ++i){
        int j = (i + 1)%n;
        int k = (j + 1)%n;
        while(j != i && k != i){
            ret = max(ret, Cross(p[j] - p[i], p[k] - p[j]));
            while(Cross(p[i] - p[j], p[(k + 1)%n] - p[k]) < 0) k = (k + 1)
                %n;
            j = (j + 1)%n;
        }
    }
    return ret;
```

```
}
```

## 5.3 半平面交 O(nlogn)

```
const double eps = 1e-6;
struct Point{
    double x, y;
    Point(double x = 0, double y = 0):x(x),y(y){}
};
typedef Point Vector;
Vector operator + (Vector A, Vector B){
    return Vector(A.x+B.x, A.y+B.y);
}
Vector operator - (Point A, Point B){
    return Vector(A.x-B.x, A.y-B.y);
}
Vector operator * (Vector A, double p){
    return Vector(A.x*p, A.y*p);
}
int sgn(double x){
    if(fabs(x) < eps)
        return 0;
    if(x < 0)
        return -1;
    return 1;
}
double Dot(Vector A, Vector B){
    return A.x*B.x + A.y*B.y;
}
double Cross(Vector A, Vector B){
    return A.x*B.y-A.y*B.x;
}
double Length(Vector A){
    return sqrt(Dot(A, A));
}
Vector Normal(Vector A){//向量A左转90°的单位法向量
    double L = Length(A);
    return Vector(-A.y/L, A.x/L);
}
```

```
36 struct Line{
37     Point p;//直线上任意一点
38     Vector v;//方向向量, 它的左边就是对应的半平面
39     double ang;//极角, 即从x轴正半轴旋转到向量v所需要的角 (弧度)
40     Line(){}
41     Line(Point p, Vector v) : p(p), v(v){
42         ang = atan2(v.y, v.x);
43     }
44     bool operator < (const Line& L) const {//排序用的比较运算符
45         return ang < L.ang;
46     }
47 };
48 //点p在有向直线L的左侧
49 bool OnLeft(Line L, Point p){
50     return Cross(L.v, p - L.p) > 0;
51 }
52 //两直线交点。假定交点唯一存在
53 Point GetIntersection(Line a, Line b){
54     Vector u = a.p - b.p;
55     double t = Cross(b.v, u)/Cross(a.v, b.v);
56     return a.p + a.v*t;
57 }
58 //半平面交的主过程
59 int HalfplaneIntersection(Line* L, int n, Point* poly){
60     sort(L, L + n);//按照极角排序
61     int fst = 0, lst = 0;//双端队列的第一个元素和最后一个元素
62     Point *P = new Point[n];//p[i] 为 q[i]与q[i + 1]的交点
63     Line *q = new Line[n];//双端队列
64     q[fst = lst = 0] = L[0];//初始化为只有一个半平面L[0]
65     for(int i = 1; i < n; ++i){
66         while(fst < lst && !OnLeft(L[i], P[lst - 1])) --lst;
67         while(fst < lst && !OnLeft(L[i], P[fst])) ++fst;
68         q[++lst] = L[i];
69         if(sgn(Cross(q[lst].v, q[lst - 1].v)) == 0){
70             //两向量平行且同向, 取内侧一个
71             --lst;
72             if(OnLeft(q[lst], L[i].p)) q[lst] = L[i];
73         }
74         if(fst < lst)
```

```
75              P[lst - 1] = GetIntersection(q[lst - 1], q[lst]);
76      }
77      while(fst < lst && !OnLeft(q[fst], P[lst - 1])) --lst;
78      //删除无用平面
79      if(lst - fst <= 1) return 0;//空集
80      P[lst] = GetIntersection(q[lst], q[fst]);//计算首尾两个半平面的交点
81      //从deque复制到输出中
82      int m = 0;
83      for(int i = fst; i <= lst; ++i) poly[m++] = P[i];
84      return m;
85  }
```

## 5.4 平面最近点对 O(nlogn)

```
1   struct Point {
2       double x,y;
3       bool operator <(const Point &a)const {
4           return x < a.x;
5       }
6   };
7   inline double dist(const Point &p1, const Point &p2) {
8       return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2
            .y));
9   }
10  Point p[maxn], q[maxn];
11  double ClosestPair(int l, int r) {
12      if(l == r)
13          return inf;
14      int mid = (l+r)>>1;
15      double tx = p[mid].x;
16      int tot = 0;
17      double ret = min(ClosestPair(l, mid), ClosestPair(mid + 1, r));
18      for(int i = l, j = mid + 1; (i <= mid || j <= r); ++i) {
19          while(j <= r && (p[i].y > p[j].y || i > mid)) {
20              q[tot++] = p[j];
21              j++; //归并按y排序
22          }
23          if(abs(p[i].x - tx) < ret && i <= mid) { //选择中间符合要求的点
24              for(int k = j - 1; k > mid && j - k < 3; --k)
```

```
25                  ret = min(ret, dist(p[i], p[k]));
26              for(int k = j; k <= r && k-j < 2; ++k)
27                  ret = min(ret, dist(p[i], p[k]));
28          }
29          if(i <= mid)
30              q[tot++] = p[i];
31      }
32      for(int i = l, j = 0; i <= r; ++i, ++j)
33          p[i] = q[j];
34      return ret;
35  }
```

## 5.5 用给定半径的圆覆盖最多的点

```
1   const int maxn = 3e2 + 5;
2   const double PI = acos(-1.0);
3   struct Point{
4       double x, y;
5   }p[maxn];
6   struct Angle{
7       double pos;
8       bool in;
9       bool operator < (const Angle& a) const {
10          return pos < a.pos;
11      }
12  }a[maxn<<1];
13  inline double Dist(Point& p1, Point& p2){
14      return sqrt((p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y))
            ;
15  }
16  //点集的大小n, 给定圆的半径r, 返回最多覆盖的点的个数
17  int solve(int n, double r){
18      int ret = 1;
19      for(int i = 0; i < n; ++i){
20          int m = 0;
21          for(int j = 0; j < n; ++j){
22              if(i == j) continue;
23              double d = Dist(p[i], p[j]);
24              if(d > 2*r) continue;
```

```
25              double alpha = atan2(p[j].y - p[i].y, p[j].x - p[i].x);
26              double beta = acos(d/(2*r));
27              a[m].pos = alpha - beta;
28              a[m++].in = true;
29              a[m].pos = alpha + beta;
30              a[m++].in = false;
31          }
32          sort(a, a + m);
33          int t = 1;
34          for(int j = 0; j < m; ++j){
35              if(a[j].in) ++t;
36              else --t;
37              if(ret < t) ret = t;
38          }
39      }
40      return ret;
41  }
```

## 5.6 给定正多边形的三个点求该正多边形的最小面积

### 5.6.1 题目大意

对于一个正多边形，只给出了其中三点的坐标，求这个多边形可能的最小面积，给出的三个点一定能够组成三角形。

### 5.6.2 思路

对于给定的三角形，其外接圆半径大小是一定的；

在外接圆半径大小一定的情况下，要使正多边形面积最小，就是使正多边形的边数最少，也就是使得每条边所对应的圆心角最大。

而最大圆心角 = 三角形三条边所对应的三个圆心角的公约数。

### 5.6.3 代码实现

```
1  const double pi = acos(-1.0);
2  const double eps = 2*pi/101;
3  struct Point{
4      double x;
5      double y;
6      Point(double x = 0, double y = 0):x(x),y(y){}
7  }p[5];
```

```
8  int sgn(double x){
9      if(fabs(x) < eps) return 0;
10     if(x > 0) return 1;
11     return -1;
12 }
13 double dist(Point a, Point b){
14     double ret = (a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y);
15     ret = sqrt(ret);
16     return ret;
17 }
18 double ang(double a, double b, double c){
19     double ret = (a*a + b*b - c*c)/(2*a*b);
20     ret = acos(ret);
21     return ret;
22 }
23 double fgcd(double a, double b){
24     if(sgn(a) == 0) return b;
25     if(sgn(b) == 0) return a;
26     return fgcd(b, fmod(a, b));
27 }
28 int main(){
29     rep(i, 0, 3) sff(p[i].x, p[i].y);
30     double a = dist(p[0], p[1]);
31     double b = dist(p[0], p[2]);
32     double c = dist(p[1], p[2]);
33 //     printf("%.6f %.6f %.6f\n", a, b, c);
34     double p = (a + b + c)/2;
35     double s = sqrt(p*(p - a)*(p - b)*(p - c));
36     double r = a*b*c/(4*s);
37 //     printf("%f\n", r);
38     double theta1 = ang(r, r, a);
39     double theta2 = ang(r, r, b);
40     double theta3 = 2*pi - theta1 - theta2;
41     double theta = fgcd(fgcd(theta1, theta2), theta3);
42 //     printf("%f\n", theta);
43     double ans = 2*pi/theta;
44     ans *= r*r*sin(theta)/2;
45     printf("%.6f\n", ans);
46     return 0;
```

```
47 }
```

## 5.7 圆上的整点

求圆心在 (0, 0) 半径为 r 的圆上的所有整点。

```
1  struct Point {
2      ll x, y; double theta;
3      Point(ll x = 0, ll y = 0):x(x), y(y) {
4          theta = atan2(x, y);
5      }
6      bool operator <(const Point& p) const {
7          return theta < p.theta;
8      }
9  };
10 struct Circle {
11     vector<Point> v;
12     void work(ll d, ll r) {
13         for(ll s = 1; s*s <= r/d; ++s) {
14             ll t = sqrt(r/d - s*s);
15             if(__gcd(s,t) == 1LL && s*s + t*t == r/d) {
16                 ll x = (s*s - t*t)/2LL*d;
17                 ll y = d*s*t;
18                 if(x > 0 && y > 0 && x*x + y*y == (r/2LL)*(r/2LL)) {
19                     v.push_back(Point(x, y)), v.push_back(Point(y, x));
20                     v.push_back(Point(x, -y)), v.push_back(Point(y, -x));
21                     v.push_back(Point(-x, y)), v.push_back(Point(-y, x));
22                     v.push_back(Point(-x, -y)), v.push_back(Point(-y, -x))
                           ;
23                 }
24             }
25         }
26     }
27     void solve(ll r) {
28         r <<= 1; v.clear();
29         for(ll i = 1; i*i <= r; ++i) {
30             if(r%i == 0) {
31                 work(i, r);
32                 if(i*i != r) work(r/i, r);
33             }
```

```
34          }
35          v.push_back(Point(r/2LL, 0)), v.push_back(Point(-r/2LL, 0));
36          v.push_back(Point(0, r/2LL)), v.push_back(Point(0, -r/2LL));
37          sort(v.begin(), v.end());
38      }
39      void pt() {
40          int sz = v.size();
41          for(int i = 0; i < sz; ++i) {
42              printf("%d: %lld %lld %f\n", i, v[i].x, v[i].y, v[i].theta);
43          }
44      }
45  } C;
```

# 6 图论

## 6.1 最小生成树（无向图）

### 6.1.1 Kruskal 算法

贪心按照边权升序排序，并查集维护块的连通性
每次贪心选择当前能选择的边权最小且不成环的边

### 6.1.2 性质

树满足以下性质：

- 是边数最多的无环图
- 是边数最少的连通图
- 任意两点间的简单路径唯一

最小生成树满足以下性质：

- 是总边权和最小的生成树
- 是最大边权最小的生成树
- 边权各不相同的图对应的最小生成树一定唯一
- 存在相同边权的最小生成树可能唯一，也可能不唯一

### 6.1.3 拓展

- 多目标规划的最小生成树只需将权值按照目标优先级依次比较即可

### 6.1.4 生成树计数（Matrix Tree 定理）

Matrix-Tree 定理 (Kirchhoff 矩阵-树定理)

- G 的度数矩阵 D[G] 是一个 n*n 的矩阵, 并且满足: 当 i != j 时,dij=0; 当 i=j 时,dij 等于 vi 的度数。
- G 的邻接矩阵 A[G] 也是一个 n*n 的矩阵, 并且满足: 如果 vi、vj 之间有边直接相连, 则 aij = 1, 否则为 0。

我们定义 G 的 Kirchhoff 矩阵 (也称为拉普拉斯算子) C[G] 为 C[G] = D[G] - A[G]

则 Matrix-Tree 定理可以描述为:G 的所有不同的生成树的个数等于其 Kirchhoff 矩阵 C[G] 任何一个 n-1 阶主子式的行列式的绝对值。

所谓 n - 1 阶主子式, 就是对于 r(1 r n), 将 C[G] 的第 r 行、第 r 列同时去掉后得到的新矩阵, 用 Cr[G] 表示。

```
struct Edge {
    int u, v, w;
} E[maxn*maxn];
ll d[maxn][maxn], a[maxn][maxn];
struct Matrix {
    ll mat[maxn][maxn];
    void init(int n) {
        for(int i = 0; i < n; ++i) {
            for(int j = 0; j < n; ++j) mat[i][j] = d[i][j] - a[i][j];
        }
    }
    ll det(int n) {
        ll res = 1;
        for(int i = 0; i < n; ++i){
            for(int j = i + 1; j < n; ++j){
                while(mat[j][i] != 0) {
                    ll t = mat[i][i]/mat[j][i]%mod;
                    for(int k = i; k < n; ++k)
                        mat[i][k] = (mat[i][k] - t*mat[j][k] + mod)%mod;
                    swap(mat[i], mat[j]);
                    res = -res;
                }
            }
            res=(res*mat[i][i])%mod;
        }
        return (res + mod)%mod;
```

```
27      }
28  } mat;
29  int n, m;
30  void init_da() {
31      for(int i = 0; i < n; ++i) {
32          for(int j = 0; j < n; ++j) {
33              d[i][j] = a[i][j] = 0;
34          }
35      }
36  }
37  void solve() {
38      scanf("%d%d", &n, &m);
39      init_da();
40      for(int ie = 1; ie <= m; ++ie) {
41          int u, v; scanf("%d%d", &u, &v);
42          ++a[u][v]; ++a[v][u];
43          ++d[u][u]; ++d[v][v];
44      }
45      mat.init(n);
46      ll ans = mat.det(n - 1);
47  }
```

### 6.1.5 最小生成树计数（Kruskal + Matrix Tree 定理）

算法引入：

给定一个含有 N 个结点 M 条边的无向图, 求它最小生成树的个数 t(G);

算法思想：

抛开“最小”的限制不看, 如果只要求求出所有生成树的个数, 是可以利用 Matrix-Tree 定理解决的;

Matrix-Tree 定理此定理利用图的 Kirchhoff 矩阵, 可以在 O(N3) 时间内求出生成树的个数;

kruskal 算法：

将图 G=V,E 中的所有边按照长度由小到大进行排序, 等长的边可以按照任意顺序;

初始化图 G’ 为 V,Ø, 从前向后扫描排序后的边, 如果扫描到的边 e 在 G’ 中连接了两个相异的连通块, 则将它插入 G’ 中;

最后得到的图 G’ 就是图 G 的最小生成树;

由于 kruskal 按照任意顺序对等长的边进行排序, 则应该将所有长度为 L0 的边的处理当作一个阶段来整体看待;

令 kruskal 处理完这一个阶段后得到的图为 G0, 如果按照不同的顺序对等长的边进行排序, 得到的 G0 也是不同;

虽然 G0 可以随排序方式的不同而不同, 但它们的连通性都是一样的, 都和 F0 的连通性相同 (F0 表示插入所有长度为 L0 的边后形成的图);

在 kruskal 算法中的任意时刻, 并不需要关注 G' 的具体形态, 而只要关注各个点的连通性如何 (一般是用并查集表示);

所以只要在扫描进行完第一阶段后点的连通性和 F0 相同, 且是通过最小代价到达这一状态的, 接下去都能找到最小生成树;

经过上面的分析, 可以看出第一个阶段和后面的工作是完全独立的;

第一阶段需要完成的任务是使 G0 的连通性和 F0 一样, 且只能使用最小的代价;

计算出这一阶段的方案数, 再乘上完成后续事情的方案数, 就是最终答案;

由于在第一个阶段中, 选出的边数是一定的, 所有边的长又都为 L0;

所以无论第一个阶段如何进行代价都是一样的, 那么只需要计算方案数就行了;

此时 Matrix-Tree 定理就可以派上用场了, 只需对 F0 中的每一个连通块求生成树个数再相乘即可;

Matrix-Tree 定理:

G 的所有不同的生成树的个数等于其 Kirchhoff 矩阵 C[G] 任何一个 n-1 阶主子式的行列式的绝对值;

n-1 阶主子式就是对于 r(1 r n), 将 C[G] 的第 r 行, 第 r 列同时去掉后得到的新矩阵, 用 Cr[G]

```
const int maxn = 2e2 + 5;
const int maxm = 2e3 + 5;
struct Edge{
    int u, v;
    int w;
    bool operator < (const Edge &ed)const{
        return w < ed.w;
    }
}edge[maxm];
int n, m;
ll mod;
int fa1[maxn], fa2[maxn]; //fa1, fa2都是并查集, fa2是每组边临时使用
bool vis[maxn];
ll g[maxn][maxn], kir[maxn][maxn]; //G顶点之间的关系, kir为生成树计数用的Kirchhoff矩阵
vector<int> v[maxn]; //记录每个连通分量
int findfa(int x, int f[]){
    int y = x;
    while(f[x] != x) x = f[x];
    while(f[y] != x){
        int z = f[y];
        f[y] = x;
        y = z;
```

```
23      }
24      return x;
25  }
26  ll det(int nn){ //生成树计数:Matrix-Tree定理
27      for(int i = 0; i < nn; ++i)
28          for(int j = 0; j < nn; ++j)
29              kir[i][j] %= mod;
30      ll ret = 1LL;
31      for(int i = 1; i < nn; ++i) {
32          for(int j = i + 1; j < nn; ++j)
33              while(kir[j][i]) {
34                  ll t = kir[i][i]/kir[j][i];
35                  for(int k = i; k < nn; ++k) kir[i][k] = (kir[i][k] - kir[j
                        ][k]*t)%mod;
36                  for(int k = i; k < nn; ++k) swap(kir[i][k], kir[j][k]);
37                  ret = -ret;
38              }
39          if(kir[i][i] == 0) return 0LL;
40          ret = (ret*kir[i][i])%mod;
41      }
42      ret = (ret + mod)%mod;
43      return ret;
44  }
45  ll Solve(){
46      sort(edge, edge + m); //按权值排序
47      for(int i = 1; i <= n; ++i){ //初始化并查集
48          fa1[i] = i;
49          vis[i] = false;
50      }
51      int wt = -1; //记录相同的权值的边
52      ll ret = 1;
53      for(int k = 0; k <= m; ++k){
54          if(k == m || edge[k].w != wt){ //一组相等的边,即权值都为Edge的边加
                完
55              for(int i = 1; i <= n; ++i){
56                  if(vis[i]){
57                      int fai = findfa(i, fa2);
58                      v[fai].push_back(i);
59                      vis[i] = false;
```

```
60                  }
61              }
62              for(int i = 1; i <= n; ++i){ //枚举每个连通分量
63                  if(v[i].size() > 1){
64                      for(int a = 1; a <= n; ++a)
65                          for(int b = 1; b <= n; ++b)
66                              kir[a][b] = 0;
67                      int len = v[i].size();
68                      for(int a = 0; a < len; ++a) //构建Kirchhoff矩阵C
69                          for(int b = a + 1; b < len; ++b){
70                              int aa = v[i][a];
71                              int bb = v[i][b];
72                              kir[b][a] -= g[aa][bb];
73                              kir[a][b] = kir[b][a];
74                              kir[a][a] += g[aa][bb]; //连通分量的度
75                              kir[b][b] += g[aa][bb];
76                          }
77                      ll now = det(len);
78                      ret = (ret*now)%mod; //对V中的每一个连通块求生成树个数
                            再相乘
79                      for(int a = 0; a < len; ++a) fa1[v[i][a]] = i;
80                  }
81              }
82              for(int i = 1; i <= n; ++i){
83                  fa1[i] = findfa(i, fa1);
84                  fa2[i] = fa1[i];
85                  v[i].clear();
86              }
87              if(k == m) break;
88              wt = edge[k].w;
89          }
90          int u = edge[k].u;
91          int v = edge[k].v;
92          int fu = findfa(u, fa1);
93          int fv = findfa(v, fa1);
94          if(fu == fv) continue;
95          vis[fu] = vis[fv] = true;
96          fa2[findfa(fu, fa2)] = findfa(fv, fa2);//并查集操作
97          ++g[fu][fv];
```

```
98          ++g[fv][fu];
99      }
100     bool flag = false;
101     for(int i = 2; i <= n && !flag; ++i) if(fa2[i] != fa2[i - 1]) flag =
            true;
102     if(m == 0) flag = true;
103     if(flag) ret = 0;
104     ret %= mod;
105     return ret;
106 }
```

## 6.2 堆优化的 Dijkstra 算法

时间复杂度 $O(n \log m)$
可进行如下拓展

- 多目标规划的最短路需：在用当前出堆的节点（当前能拓展的最近点），更新其他相邻节点时权值按照目标优先级依次比较，决定是否将目标入堆
- 最短路径数量统计需：在某个节点最短距离被缩短时重置为当前节点的方案数，当前距离相同时，累加当前节点的方案数
- 维护次短路需：当最短距离可以被松弛时，原最短距离更新为现最短距离，次短距离可更新为原最短距离；否则若能松弛次短距离就松弛（类似于多目标规划）

## 6.3 SPFA O(玄学) 判负权环

### 6.3.1 模板

```
1  const int maxn = 1e5 + 5;
2  int n;
3  const int inf = 0x3f3f3f3f;
4  struct node{
5      int to, val;
6      node(int to = 0, int val = 0):to(to), val(val){}
7  };
8  bool inq[maxn];
9  int dist[maxn], cnt[maxn];
10 vector<node> g[maxn];
11 queue<int> q;
12 bool spfa(int s) {
13     for (int i = 1; i <= n; ++i) {
```

```
14          inq[i] = false;
15          dist[i] = inf;
16          cnt[i] = 0;
17      }
18      q.push(s); dist[s] = 0;
19      inq[s] = true; ++cnt[s];
20      while (!q.empty()){
21          int x = q.front(); q.pop();
22          inq[x] = false;
23          int sz = g[x].size();
24          for (int i = 0; i < sz; ++i) {
25              node now = g[x][i];
26              int nv = dist[x] + now.val;
27              if (dist[now.to] > nv){
28                  dist[now.to] = nv;
29                  if (!inq[now.to]) {
30                      inq[now.to] = true;
31                      q.push(now.to);
32                      if (++cnt[now.to] >= n) return false;
33                  }
34              }
35          }
36      }
37      return true;
38  }
```

### 6.3.2 差分约束系统

$X_a - X_b \leq c$

n: number of variables 变量数

m: number of constraint conditions 约束条件数

Super Source 0: $X_i - X_0 \leq 0$ 超级源点 0

算法运行结束后，若无解则返回 false

否则，dist 数组里为一组非正整数解

```
1   const int maxn = 5e5 + 5;
2   const ll linf = 0x3f3f3f3f3f3f3f3f;
3   struct node{
4       int to; ll val;
5       node(int to = 0, ll val = 0):to(to), val(val){}
```

```
};
bool inq[maxn];
ll dist[maxn];
int cnt[maxn], n, m;
vector<node> g[maxn];
queue<int> q;
bool spfa(int s){
    for(int i = 0; i <= n; ++i){
        inq[i] = false;
        dist[i] = linf;
        cnt[i] = 0;
    }
    q.push(s); dist[s] = 0;
    inq[s] = true; ++cnt[s];
    while(!q.empty()){
        int x = q.front(); q.pop();
        int sz = g[x].size();
        for(int i = 0; i < sz; ++i){
            node now = g[x][i];
            if(now.to == x && now.val < 0) return false;
            ll nv = dist[x] + now.val;
            if(dist[now.to] > nv){
                dist[now.to] = nv;
                if(!inq[now.to]){
                    inq[now.to] = true;
                    q.push(now.to);
                    if(++cnt[now.to] >= n) return false;
                }
            }
        }
        inq[x] = false;
    }
    return true;
}
int a[maxn], b[maxn]; ll c[maxn];
bool DCS() {
    for(int i = 1; i <= n; ++i) g[0].push_back(node(i, 0));
    for(int i = 0; i < m; ++i) g[b[i]].push_back(node(a[i], c[i]));
    if(spfa(0)) return true;
```

```
45      return false;
46  }
```

## 6.4 强连通分量（Tarjan 算法）

### 6.4.1 无向图

```
1   struct Node {
2       int next, v;
3   } star[maxn<<1];
4   int head[maxn], tot;
5   int dfn[maxn], low[maxn], vis[maxn], cnt;
6   int s[maxn], color[maxn], top, col;
7   int pre[maxn];
8
9   void init(){
10      tot = 0, cnt = 0;
11      col = 0, top = -1;
12      memset(vis, 0, sizeof(vis));
13      memset(pre, 0, sizeof(vis));
14      memset(head, -1, sizeof(vis));
15  }
16
17  void addedge(int u, int v){
18      star[tot].next = head[u];
19      star[tot].v = v;
20      head[u] = tot++;
21  }
22
23  void Tarjan(int u){
24      vis[u] = true;
25      dfn[u] = low[u] = cnt++;
26      s[++top] = u;
27      for(int i = head[u]; ~i; i = star[i].next){
28          int v = star[i].v;
29          if(!vis[v]) {
30              pre[v] = u;
31              Tarjan(v);
32          }
33          if(v != pre[u]) low[u] = min(low[u], low[v]);
```

```
34      }
35
36      if(dfn[u] == low[u]){
37          ++col;
38          do{
39              color[s[top]] = col;
40          }while(top != -1 && s[top--] != u);
41      }
42  }
```

### 6.4.2 普通有向图转 DAG 全家桶（去重边，去自环）

```
1   const int maxn = 1e5 + 5;
2   struct Gragh {
3       struct Edge {
4           int to, nxt;
5       } E[maxn << 1];
6       int head[maxn], tot;
7       void init(int n) {
8           for (int i = 0; i <= n; ++i) head[i] = -1;
9           tot = 0;
10      }
11      void add_edge(int u, int v) {
12          E[tot] = {v, head[u]};
13          head[u] = tot++;
14      }
15  } G1, G2;
16  int dfn[maxn], low[maxn], cnt;
17  int st[maxn], tp, color[maxn];
18  bool ins[maxn];
19  int n, m, t, N;
20  void Tarjan(int x) {
21      ins[x] = true;
22      dfn[x] = low[x] = ++cnt;
23      st[tp++] = x;
24      for (int i = G1.head[x]; i != -1; i = G1.E[i].nxt) {
25          int to = G1.E[i].to;
26          if (dfn[to] == 0) {
27              Tarjan(to);
```

```
28              low[x] = min(low[x], low[to]);
29          } else if (ins[to]) {
30              low[x] = min(low[x], dfn[to]);
31          }
32      }
33      if (dfn[x] == low[x]) {
34          ++N;
35          while (true) {
36              int now = st[--tp];
37              color[now] = N; ins[now] = false;
38              // maintain something
39              if (now == x) break;
40          }
41      }
42  }
43  unordered_map<int, int> vis[maxn];
44  void solve() {
45      cnt = tp = N = 0;
46      for(int i = 0; i <= n; ++i) {
47          ins[i] = false;
48          dfn[i] = low[i] = 0;
49      }
50      for(int i = 1; i <= n; ++i) {
51          if(dfn[i] != 0) continue;
52          Tarjan(i);
53      }
54      for(int i = 1; i <= n; ++i) {
55          for(int j = G1.head[i]; j != -1; j = G1.E[j].nxt) {
56              int to = G1.E[j].to;
57              int ci = color[i], ct = color[to];
58              if(ci == ct) continue;
59              ++vis[ci][ct];
60              if(vis[ci][ct] == 1) continue;
61              G2.add_edge(ci, ct);
62          }
63      }
64  }
```

### 6.4.3 在 DAG 的拓扑序上动态规划（BZOJ-1179 Atm）

1 号点出发，单向边所构成的图，每个点有点权，经过某个点一次和多次只会获得一次该点的点权的收益，问最大能够获得多大收益。

```
const int maxn = 5e5 + 5;
struct Edge {
    int to, nxt;
} E1[maxn << 1], E2[maxn << 1];
int head1[maxn], head2[maxn], tot1, tot2;
int n, m;
void init() {
    for(int i = 0; i <= n; ++i) {
        head1[i] = head2[i] = -1;
    }
    tot1 = tot2 = 0;

}
void addEdge1(int u, int v) {
    E1[tot1] = {v, head1[u]};
    head1[u] = tot1++;
}
void addEdge2(int u, int v) {
    E2[tot2] = {v, head2[u]};
    head2[u] = tot2++;
}
int dfn[maxn], low[maxn], cnt;
int st[maxn], color[maxn], tp, N;
bool ins[maxn], ok1[maxn], ok2[maxn];
int val1[maxn], val2[maxn];
void Tarjan(int x) {
    ins[x] = true;
    dfn[x] = low[x] = ++cnt;
    st[tp++] = x;
    for(int i = head1[x]; i != -1; i = E1[i].nxt) {
        int to = E1[i].to;
        if(dfn[to] == 0) {
            Tarjan(to);
            low[x] = min(low[x], low[to]);
        } else if(ins[to]) {
            low[x] = min(low[x], dfn[to]);
```

```
37          }
38      }
39      if(dfn[x] == low[x]) {
40          ++N;
41          while(true) {
42              int now = st[--tp];
43              color[now] = N; ins[now] = false;
44              //    这里根据题目要求维护相关信息即可
45              ok2[N] |= ok1[now];
46              val2[N] += val1[now];
47              if(now == x) break;
48          }
49      }
50  }
51  unordered_map<int, bool> vis[maxn];
52  void Solve() {
53      cnt = tp = N = 0;
54      for(int i = 0; i <= n; ++i) {
55          ins[i] = false;
56          dfn[i] = low[i] = val2[i] = 0;
57      }
58      for(int i = 1; i <= n; ++i) {
59          if(dfn[i] != 0) continue;
60          Tarjan(i);
61      }
62      for(int i = 1; i <= n; ++i) {
63          for(int j = head1[i]; j != -1; j = E1[j].nxt) {
64              int to = E1[j].to;
65              int ci = color[i], ct = color[to];
66              if(ci == ct) continue;
67              if(vis[ci][ct]) continue;
68              vis[ci][ct] = true;
69              addEdge2(ci, ct);
70          }
71      }
72  }
73  int sum[maxn], q[maxn], ind2[maxn], lf, rt;
74  void Dfs(int x) {
75      for(int i = head2[x]; i != -1; i = E2[i].nxt) {
```

```
76          int to = E2[i].to;
77          if(ind2[to] == 0) Dfs(to);
78          ++ind2[to];
79      }
80  }
81  void Topo(int s) {
82      lf = rt = 0;
83      for(int i = 1; i <= N; ++i) ind2[i] = 0;
84      Dfs(s);
85      q[rt++] = s;
86      while(lf < rt) {
87          int x = q[lf++];
88          sum[x] += val2[x];
89          for(int i = head2[x]; i != -1; i = E2[i].nxt) {
90              int to = E2[i].to;
91              sum[to] = max(sum[to], sum[x]);
92              if(--ind2[to] == 0) q[rt++] = to;
93          }
94      }
95  }
96  int main() {
97      scanf("%d%d", &n, &m);
98      init();
99      for(int i = 0; i < m; ++i) {
100         int x, y; scanf("%d%d", &x, &y);
101         addEdge1(x, y);
102     }
103     for(int i = 1; i <= n; ++i) scanf("%d", &val1[i]);
104     int s, p; scanf("%d%d", &s, &p);
105     for(int i = 0; i < p; ++i) {
106         int x; scanf("%d", &x);
107         ok1[x] = true;
108     }
109     Solve();
110     for(int i = 0; i <= n; ++i) sum[i] = 0;
111     Topo(color[s]);
112     int ans = 0;
113     for(int i = 1; i <= n; ++i) {
114         if(!ok2[i]) continue;
```

```
115             ans = max(ans, sum[i]);
116         }
117         printf("%d\n", ans);
118         return 0;
119 }
```

## 6.5 Cayley 公式（无根树计数）

### 6.5.1 公式内容

对于任何正整数 $n$

$n$ 个节点的带标号的无根树有 $n^{n-2}$ 个

还有一种等价的说法：

$n$ 个有标号节点的无向完全图的生成树有 $n^{n-2}$ 棵

### 6.5.2 限制特定节点的度的树的计数

限制编号为 $i$ 的点的度数为 $d_i$，则树的个数有

$$\frac{(n-2)!}{\Pi_{i=1}^{n}(d_i-1)!}$$

### 6.5.3 n 个节点 k 棵树的森林计数问题

令 $T_{n,k}$ 表示有着 $n$ 个节点的 $k$ 个联通块（$k$ 棵树）的森林的种类数，使得 $1,2,3,\cdots,k$ 属于不同的子树

根据 Cayley 定理，可以推导出

$$T_{n,k} = kn^{n-k-1}$$

例题：BZOJ 1005 [HNOI2008] 明明的烦恼

给出标号为 1 到 N （$0 < N \leq 1000$）的点，以及某些点最终的度数 $D_i$（如果对度数不要求，则输入 $-1$）

允许在任意两点间连线，可产生多少棵度数满足要求的树?

根据 Cayley 公式的推广式一

$$\frac{(n-2)!}{\Pi_{i=1}^{n}(d_i-1)!}$$

如果这道题目没有不受限制的点，那么这道问题就结束啦

可是题目里面有不受限制的点，我们应该如何处理呢

假设度数有限制的点的数量为 cnt，它们的度数为 $D_i$

令

$$sum = \sum_{i=1}^{cnt}(D_i - 1)$$

那么，在序列中不同的排列的总数为：

$$C_{n-2}^{sum} \times \frac{sum!}{\Pi_{i=1}^{cnt}(D_i - 1)!}$$

而剩下的 $n-2-sum$ 个位置，可以随意的排列剩余的 $n-cnt$ 个点，于是，总的方案数就应该是：

$$C_{n-2}^{sum} \times \frac{sum!}{\Pi_{i=1}^{cnt}(D_i - 1)!} \times (n-cnt)^{n-2-sum}$$

化简后得到：

$$\frac{(n-2)!}{(n-2-sum)\Pi_{i=1}^{cnt}(D_i - 1)!} \times (n-cnt)^{n-2-sum}$$

注意答案会很大，并且没有取模操作

所以需要用到大整数进行运算

代码实现

```
import java.math.BigInteger;
import java.util.Scanner;
public class Main {
    static int d[] = new int[1005];
    static BigInteger fac[] = new BigInteger[1005];
    public static void main(String[] args) {
    // write your code here
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        int sum = 0, cnt = 0;
        boolean f = false;
        for(int i = 0; i < n; ++i){
            d[i] = in.nextInt();
            if(d[i] == 0 || d[i] >= n) f = true;
            if(d[i] == -1) continue;
            sum += d[i] - 1;
            ++cnt;
        }
        in.close();
        if(n == 1){
            if(!f) System.out.println(1);
            else System.out.println(0);
            return ;
```

```
24          }
25          if(n == 2){
26              if(f || d[0] == 0 || d[1] == 0) System.out.println(0);
27              else System.out.println(1);
28              return ;
29          }
30          if(f){
31              System.out.println(0);
32              return ;
33          }
34          fac[0] = BigInteger.valueOf(1);
35          for(int i = 1; i <= n; ++i) fac[i] = fac[i - 1].multiply(
                BigInteger.valueOf(i));
36          BigInteger ans = fac[n - 2].divide(fac[n - 2 - sum]);
37          for(int i = 0; i < n; ++i){
38              if(d[i] == -1) continue;
39              ans = ans.divide(fac[d[i] - 1]);
40          }
41          for(int i = 0; i < n - 2 - sum; ++i) ans = ans.multiply(BigInteger
                .valueOf(n - cnt));
42          System.out.println(ans);
43      }
44  }
```

## 6.6 树上倍增求 LCA

```
1   struct edge{
2       int to; ll w;
3       edge(int to = 0, ll w = 0):to(to), w(w){}
4   };
5   vector<edge> g[maxn];
6   int fa[maxn][maxm], d[maxn];
7   ll dist[maxn], mxx[maxn][maxm];
8   void addedge(int u, int v, ll w){
9       g[u].push_back(edge(v, w));
10      g[v].push_back(edge(u, w));
11  }
12  void init(int n){
13      for(int i = 0; i <= n; ++i){
```

```
14          g[i].clear();
15          for(int j = 0; j < maxm; ++j){
16              mxx[i][j] = 0;
17              fa[i][j] = 0;
18          }
19      }
20  }
21  void dfs(int x, int y){
22      for(int i = 1; i < maxm; ++i){
23          mxx[x][i] = max(mxx[fa[x][i - 1]][i - 1], mxx[x][i - 1]);
24          fa[x][i] = fa[fa[x][i - 1]][i - 1];
25      }
26      int sz = g[x].size();
27      for(int i = 0; i < sz; ++i){
28          int to = g[x][i].to;
29          ll w = g[x][i].w;
30          if(to == y) continue;
31          d[to] = d[x] + 1;
32          dist[to]  = dist[x] + w;
33          fa[to][0] = x;
34          mxx[to][0] = w;
35          dfs(to, x);
36      }
37  }
38
39  ll query(int a, int b){
40      int x = a, y = b;
41      int lca;
42      ll len, mx = 0;
43      if(d[a] > d[b]) swap(a, b);
44      int dt = d[b] - d[a];
45      for(int i = 0; (1<<i) <= dt; i++){
46          if(((dt>>i)&1) == 0) continue;
47          mx = max(mx, mxx[b][i]);
48          b = fa[b][i];
49      }
50      for(int i = maxm - 1; i >= 0; --i){
51          if(fa[a][i] == fa[b][i]) continue;
52          mx = max(mx, max(mxx[a][i], mxx[b][i]));
```

```
53          a = fa[a][i]; b = fa[b][i];
54      }
55      if(a == b){
56          lca = a;
57      }
58      else{
59          lca = fa[a][0];
60          mx = max(mx, max(mxx[a][0], mxx[b][0]));
61      }
62      len = dist[x] + dist[y] - 2LL*dist[lca];
63      return mx; // len or lca or mx or mi
64  }
```

## 6.7 树上点分治（树上合法点对（路径）计数）

### 6.7.1 写法 1（使用 id 标记来自哪一棵子树去重）

给一颗 n 个节点的树，每条边上有一个距离 v。

给定 k 值，求有多少点对（u，v）使 u 到 v 的距离小于等于 k。

```
1  #include <cstdio>
2  #include <algorithm>
3  using namespace std;
4  typedef long long ll;
5  const ll mod = 1e9 + 7;
6  const int maxn = 1e4 + 5;
7  const int inf = 0x3f3f3f3f;
8  struct Edge {
9      int nxt, to; ll w;
10     Edge(int nxt = 0, int to = 0, ll w = 0):nxt(nxt), to(to), w(w) {}
11 } E[maxn << 1];
12 int tot, head[maxn];
13 int n, root, mi, nsz, atot, sz[maxn];
14 bool vis[maxn];
15 ll k, ans, cnt[maxn];
16 void init() {
17     for(int i = 0; i <= n; ++i) {
18         head[i] = -1; cnt[i] = 0;
19         vis[i] = false;
20     }
21     tot = 0; ans = 0;
```

```
}
void addEdge(int u, int v, ll w) {
    E[tot] = Edge(head[u], v, w);
    head[u] = tot++;
}
void GetRoot(int x, int y) {
    sz[x] = 1; int mxs = 0;
    for(int i = head[x]; i != -1; i = E[i].nxt) {
        int to = E[i].to;
        if(vis[to] || to == y) continue;
        GetRoot(to, x);
        sz[x] += sz[to];
        if(sz[to] > mxs) mxs = sz[to];
    }
    if(nsz - sz[x] > mxs) mxs = nsz - sz[x];
    if(mxs < mi) mi = mxs, root = x;
}
struct node {
    int id; ll w;
    bool operator <(const node& nd) const {
        return w < nd.w;
    }
    node(int id = 0, ll w = 0):id(id), w(w) {}
} a[maxn];
void dfs(int x, int y, int id, ll w) {
    a[atot++] = node(id, w);
    for(int i = head[x]; i != -1; i = E[i].nxt) {
        int to = E[i].to; ll nw = E[i].w;
        if(vis[to] || to == y) continue;
        dfs(to, x, id, w + nw);
    }
}
void GetA(int x) {
    a[atot++] = node(x, 0);
    for(int i = head[x]; i != -1; i = E[i].nxt) {
        int to = E[i].to; ll w = E[i].w;
        if(vis[to]) continue;
        dfs(to, x, to, w);
    }
```

```
61 }
62 ll calc(int x) {
63     ll ret = 0;
64     atot = 0; GetA(x);
65     int rt = atot - 1;
66     sort(a, a + atot);
67     for(int i = 0; i < atot; ++i) ++cnt[a[i].id];
68     for(int lf = 0; ; ++lf) {
69         --cnt[a[lf].id];
70         while(lf < rt && a[lf].w + a[rt].w > k) {
71             --cnt[a[rt].id];
72             --rt;
73         }
74         if(lf >= rt) break;
75         ret += rt - lf - cnt[a[lf].id];
76     }
77     return ret;
78 }
79 void solve(int x) {
80     ans += calc(x); vis[x] = true;
81     int tsz = nsz;
82     for(int i = head[x]; i != -1; i = E[i].nxt) {
83         int to = E[i].to; ll w = E[i].w;
84         if(vis[to]) continue;
85         mi = inf;
86         if(sz[to] > sz[x]) nsz = tsz - sz[x];
87         else nsz = sz[to];
88         GetRoot(to, -1); solve(root);
89     }
90 }
91 int main(){
92     while(scanf("%d%lld", &n, &k) != EOF) {
93         if(n == 0 && k == 0) break;
94         init();
95         for(int i = 1; i < n; ++i) {
96             int u, v; ll w; scanf("%d%d%lld", &u, &v, &w);
97             addEdge(u, v, w); addEdge(v, u, w);
98         }
99         nsz = n, mi = inf;
```

```
100         GetRoot(1, -1); solve(root);
101         printf("%lld\n", ans);
102     }
103     return 0;
104 }
```

### 6.7.2 写法 2（容斥去重）

给定一棵树，有 m 次询问，每次询问存不存在两个点的距离为 k

```
1  #include <cstdio>
2  #include <algorithm>
3  using namespace std;
4  typedef long long ll;
5  const ll mod = 1e9 + 7;
6  const int maxn = 1e4 + 5;
7  const int inf = 0x3f3f3f3f;
8  struct Edge {
9      int nxt, to; ll w;
10     Edge(int nxt = 0, int to = 0, ll w = 0):nxt(nxt), to(to), w(w) {}
11 } E[maxn << 1];
12 int tot, head[maxn];
13 int n, root, mi, nsz, atot, sz[maxn];
14 bool vis[maxn];
15 ll k, ans;
16 void init() {
17     for(int i = 0; i <= n; ++i) head[i] = -1;
18     tot = 0;
19 }
20 void addEdge(int u, int v, ll w) {
21     E[tot] = Edge(head[u], v, w);
22     head[u] = tot++;
23 }
24 void GetRoot(int x, int y) {
25     sz[x] = 1; int mxs = 0;
26     for(int i = head[x]; i != -1; i = E[i].nxt) {
27         int to = E[i].to;
28         if(vis[to] || to == y) continue;
29         GetRoot(to, x);
30         sz[x] += sz[to];
```

```
31          if(sz[to] > mxs) mxs = sz[to];
32      }
33      if(nsz - sz[x] > mxs) mxs = nsz - sz[x];
34      if(mxs < mi) mi = mxs, root = x;
35  }
36  ll a[maxn];
37  void dfs(int x, int y, ll w) {
38      a[atot++] = w;
39      for(int i = head[x]; i != -1; i = E[i].nxt) {
40          int to = E[i].to; ll nw = E[i].w;
41          if(vis[to] || to == y) continue;
42          dfs(to, x, w + nw);
43      }
44  }
45
46  ll calc(int x, ll w) {
47      ll ret = 0, lst = 0;
48      atot = 0; dfs(x, -1, w);
49      sort(a, a + atot);
50      int lf = 0, rt = atot - 1;
51      while(lf < rt) {
52          ll sum = a[lf] + a[rt];
53          if(sum > k) --rt;
54          else if(sum < k) ++lf;
55          else {
56              if(a[lf] == a[rt]) {
57                  ll len = rt - lf + 1;
58                  ret += (len - 1LL)*len/2LL;
59                  break;
60              }
61              int pl = lf, pr = rt;
62              while(pl < rt && a[pl] == a[lf]) ++pl;
63              while(pr > lf && a[pr] == a[rt]) --pr;
64              ll lenl = pl - lf, lenr = rt - pr;
65              ret += lenl*lenr;
66              lf = pl, rt = pr;
67          }
68      }
69      return ret;
```

```
70  }
71  void solve(int x) {
72      ans += calc(x, 0); vis[x] = true;
73      int tsz = nsz;
74      for(int i = head[x]; i != -1; i = E[i].nxt) {
75          int to = E[i].to; ll w = E[i].w;
76          if(vis[to]) continue;
77          ans -= calc(to, w);
78          mi = inf;
79          if(sz[to] > sz[x]) nsz = tsz - sz[x];
80          else nsz = sz[to];
81          GetRoot(to, -1); solve(root);
82      }
83  }
84  int main(){
85      while(scanf("%d", &n) != EOF) {
86          if(n == 0) break;
87          init();
88          for(int i = 1; i <= n; ++i) {
89              int x;
90              while(scanf("%d", &x) != EOF) {
91                  if(x == 0) break;
92                  ll y; scanf("%lld", &y);
93                  addEdge(i, x, y);
94                  addEdge(x, i, y);
95              }
96          }
97          ll val;
98          while(scanf("%lld", &val) != EOF) {
99              if(val == 0) break;
100             for(int i = 1; i <= n; ++i) vis[i] = false;
101             k = val; ans = 0;
102             nsz = n, mi = inf;
103             GetRoot(1, -1); solve(root);
104             if(ans > 0) puts("AYE");
105             else puts("NAY");
106         }
107         puts(".");
108     }
```

```
109     return 0;
110 }
```

## 6.8 输出欧拉回路（路径）

```
1  #include <iostream>
2  #include <cstdio>
3  #include <cmath>
4  #include <bits/stdc++.h>
5
6  using namespace std;
7  typedef long long ll;
8  const int mod = 1e9 + 7;
9  const int inf = 0x3f3f3f3f;
10
11 #pragma comment(linker, "/STACK:102400000, 102400000")
12 const int maxn = 1e5 + 5;
13 const int maxm = 2e5 + 15;
14 struct Edge {
15     int to, nxt;
16 } E[maxm << 1];
17 int head[maxn], te;
18 int road[maxm], tr;
19 int vertex[maxm], tv;
20 bool vis[maxm];
21 void init() {
22     memset(head, -1, sizeof(head));
23     te = 1;
24 }
25 int ind[maxn], outd[maxn];
26 void addEdge(int u, int v, bool undir) {
27     E[te << 1] = {v, head[u]};
28     head[u] = (te << 1);
29     ++outd[u], ++ind[v];
30     if(undir) {
31         E[te << 1 | 1] = {u, head[v]};
32         head[v] = (te << 1 | 1);
33         ++outd[v], ++ind[u];
34     }
```

```
35      ++te;
36  }
37  void dfs(int x) {
38      for(int &i = head[x]; i != -1; i = E[i].nxt) { // 对 head[x] 的引用是
            降低复杂度关键
39          int id = (i >> 1), ii = i;
40          if(vis[id]) continue;
41          vis[id] = true;
42          dfs(E[ii].to);
43          if((ii & 1) == 0) road[tr++] = id;
44          else road[tr++] = -id;
45          if(i == -1) break;
46      }
47      vertex[tv++] = x;
48  }
49  int n, m;
50  bool Euler(bool undir) {
51      int id = E[2].to;
52      for(int i = 1; i <= n; ++i) {
53          if(!undir && ind[i] != outd[i]) return false; // 有向图
54          if(undir && (ind[i] & 1) == 1) return false; // 无向图
55      }
56      tv = tr = 0;
57      dfs(id);
58      if(tr < m) return false;
59      return true;
60  }
61  int main() {
62      int ty; scanf("%d", &ty);
63      scanf("%d%d", &n, &m);
64      init();
65      memset(vis, false, sizeof(vis));
66      memset(ind, 0, sizeof(ind));
67      memset(outd, 0, sizeof(outd));
68      bool undir = true;
69      if(ty == 1) undir = true; // 无向图
70      else undir = false; // 有向图
71      for(int i = 0; i < m; ++i) {
72          int u, v;
```

```
73          scanf("%d%d", &u, &v);
74          addEdge(u, v, undir);
75      }
76      if(Euler(undir)) {
77          puts("YES");
78          for(int i = tr - 1; i >= 0; --i) {
79              if(i < tr - 1) printf(" ");
80              printf("%d", road[i]);
81          }
82          puts("");
83      } else {
84          puts("NO");
85      }
86      return 0;
87  }
```

## 6.9 多源最短路（floyd 算法）

### 6.9.1 无向图最小环

```
1   const int inf = 0x3f3f3f3f;
2   const int maxn = 1e2 + 5;
3   int g[maxn][maxn], dist[maxn][maxn], n, m;
4   int MincostCycle() {
5       int ret = inf;
6       for(int k = 1; k <= n; ++k) {
7           for(int i = 1; i < k; ++i) {
8               if(g[i][k] == inf) continue;
9               for(int j = 1; j < k; ++j) {
10                  if(i == j) continue;
11                  if(g[k][j] == inf) continue;
12                  if(dist[j][i] == inf) continue;
13                  ret = min(ret, dist[j][i] + g[i][k] + g[k][j]);
14              }
15          }
16          for(int i = 1; i <= n; ++i) {
17              if(dist[i][k] == inf) continue;
18              for(int j = 1; j <= n; ++j) {
19                  if(dist[k][j] == inf) continue;
20                  dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
```

```
21                }
22            }
23        }
24        return ret;
25    }
```

### 6.9.2 输出路径

迭代输出字典序最小的路径

```
1   const int inf = 0x3f3f3f3f;
2   const int maxn = 1e2 + 5;
3   int g[maxn][maxn], b[maxn], n;
4   int path[maxn][maxn];
5   int way[maxn], tot;
6   void GetPath(int a, int b) {
7       while(a != b) {
8           a = path[a][b];
9           way[tot++] = a;
10      }
11  }
12  void Floyd() {
13      for(int k = 1; k <= n; ++k) {
14          for(int i = 1; i <= n; ++i) {
15              if(i == k) continue;
16              if(g[i][k] == inf) continue;
17              for(int j = 1; j <= n; ++j) {
18                  if(i == j || j == k) continue;
19                  if(g[k][j] == inf) continue;
20                  int w = g[i][k] + g[k][j] + b[k];
21                  if(w < g[i][j]) {
22                      g[i][j] = w;
23                      path[i][j] = path[i][k];
24                  } else if(w == g[i][j] && path[i][k] < path[i][j]) {
25                      path[i][j] = path[i][k];
26                  }
27              }
28          }
29      }
30  }
```

## 6.10 三元环计数

求 $N$ 个点，$M$ 条边的无向图中，三元环的个数

将点按照度数为第一关键字，标号为第二关键字从小到大排序，定义排序后每个点的序为 $pos[x]$

对于每条无线边，定向为：pos 小的点连向 pos 大的点

这样连完之后，每个点的出度最多只有 $\sqrt{M}$

然后，依 pos 从小到大枚举验证三元组即可，总时间复杂度 $O(M\sqrt{M})$

```
typedef pair<int, int> pii;
const int maxn = 1e5 + 5;
const int maxm = 2e5 + 5;
struct Edge {
    int to, nxt;
} E[maxm];
int head[maxn], tot;
int u[maxm], v[maxm]; // input: id \in [0, n)
int cnt[maxm]; // cnt[i]: how many triples the i-th edge involve in
int ind[maxn], vis[maxn], frm[maxn];
void init_edge(int n) {
    for(int i = 0; i <= n; ++i) {
        head[i] = -1;
    }
    tot = 0;
}
void add_edge(int u, int v) {
    E[tot] = {v, head[u]};
    head[u] = tot++;
}
int cmp(const int &x, const int &y) {
    if(ind[x] == ind[y]) {
        return x - y;
    }
    return ind[x] - ind[y];
}
void get_triples(int n, int m) {
    init_edge(n);
    for(int i = 1; i <= n; ++i) {
        ind[i] = 0; vis[i] = 0;
    }
    for(int i = 0; i < m; ++i) {
        cnt[i] = 0;
```

```
34          ++ind[u[i]]; ++ind[v[i]];
35      }
36      for(int i = 0; i < m; ++i) {
37          const int x = u[i], y = v[i];
38          if(x == y) { // self-loop: depends with the problem
39              // assert(false);
40              continue;
41          }
42          if(cmp(x, y) > 0) {
43              add_edge(x, y);
44          } else {
45              add_edge(y, x);
46          }
47      }
48      for(int i = 1; i <= n; ++i) {
49          const int x = i;
50          for(int j = head[x]; j != -1; j = E[j].nxt) {
51              const int y = E[j].to;
52              vis[y] = x; frm[y] = j;
53          }
54          for(int j = head[x]; j != -1; j = E[j].nxt) {
55              const int y = E[j].to;
56              for(int k = head[y]; k != -1; k = E[k].nxt) {
57                  const int z = E[k].to;
58                  if(vis[z] == x) {
59                      // find a triple edge(a, b, c)
60                      const int a = j;
61                      const int b = k;
62                      const int c = frm[z];
63                      ++cnt[a]; ++cnt[b]; ++cnt[c];
64                  }
65              }
66          }
67      }
68      // calculate something with array: cnt[]
69  }
```

# 7 网络流与匹配问题

## 7.1 二分图匹配

### 7.1.1 KM 算法 $O(n^3)$

```
const int maxn = 2e2 + 10 ; // 点的个数
const int inf = 0x3f3f3f3f;
int lx[maxn], ly[maxn], slack[maxn];
int match[maxn];
int g[maxn][maxn];
int X, Y; // X点集大小, Y点集大小
bool visx[maxn], visy[maxn];
void init(){
    for(int i = 1; i <= X; i++) // 根据X点集点的编号修改循环起始
        for(int j = 1; j <= Y; j++) // 根据Y点集点的编号修改循环起始
            g[i][j] = -inf; // 根据题意更改初始化值
    return ;
}
// 调用初始化函数后读图, 由于使用邻接矩阵存图, 根据题意判重边, 处理重边信息
int path(int x){
    visx[x] = true;
    for(int i = 1; i <= Y; i++) {
        if(visy[i])
            continue;
        int tmp = lx[x] + ly[i] - g[x][i];
        if(tmp == 0){
            visy[i] = true;
            if(match[i] == -1 || path(match[i])){
                match[i] = x;
                return true;
            }
        }
        else
            slack[i] = min(slack[i], tmp);
    }
    return false;
}
int KM(){ // 返回二分图权值和最大的完备匹配
    memset(ly, false, sizeof(ly));
```

```
35      memset(match, -1, sizeof(match));
36      for(int i = 1; i <= X; i++) { // 根据X点集点的编号修改循环起始
37          lx[i] = -inf;
38          for(int j = 1; j <= Y; j++){ // 根据Y点集点的编号修改循环起始
39              lx[i] = max(lx[i], g[i][j]);
40          }
41      }
42      for(int i = 1; i <= X; i++) { // 根据X点集点的编号修改循环起始
43          memset(slack, inf, sizeof(slack));
44          while(true) {
45              memset(visx, false, sizeof(visx));
46              memset(visy, false, sizeof(visy));
47              if(path(i))
48                  break;
49              int d = inf;
50              for(int j = 1; j <= Y; j++){ // 根据Y点集点的编号修改循环起始
51                  if(!visy[j])
52                      d = min(d, slack[j]);
53              }
54              for(int j = 1; j <= Y; j++){ // 根据Y点集点的编号修改循环起始
55                  if(visx[j])
56                      lx[j] -= d ;
57              }
58              for(int j = 1; j <= Y; j++){ // 根据Y点集点的编号修改循环起始
59                  if(visy[j])
60                      ly[j] += d;
61                  else
62                      slack[j] -=d;
63              }
64          }
65      }
66      int ans = 0;
67      for(int i = 1; i <= Y; i++) // 根据Y点集点的编号修改循环起始
68          ans += g[match[i]][i]; // 注意根据是否可以完全匹配加判断语句
69      return ans;
70  }
```

### 7.1.2 HK 算法（非稠密图 $O(n\sqrt{n})$）

```
const int maxn = 2e5 + 5;
int Nl, Nr, now;
int matchl[maxn], matchr[maxn], levell[maxn], levelr[maxn], visited[maxn];
vector<int> g[maxn];
int q[maxn];
bool bfs() {
    int* begin(q);
    int* end(q);
    bool ret = false;
    for (int i = 1; i <= Nl; ++i) {
        if (matchl[i]) {
            levell[i] = 0;
        } else {
            levell[i] = 1;
            *end++ = i;
        }
    }
    for (int i = 1; i <= Nr; ++i) levelr[i] = 0;
    while (begin != end) {
        int u = (*begin++);
        int sz = g[u].size();
        for (int i = 0; i < sz; ++i) {
            int to = g[u][i];
            if (levelr[to] == 0) {
                levelr[to] = levell[u] + 1;
                if (matchr[to]) {
                    levell[matchr[to]] = levelr[to] + 1;
                    *end++ = matchr[to];
                }
                else
                    ret = true;
            }
        }
    }
    return ret;
}
int dfs(int u) {
    int sz = g[u].size();
    for (int i = 0; i < sz; ++i) {
```

```
40          int to = g[u][i];
41          if (levelr[to] == levell[u] + 1 && visited[to] != now) {
42              visited[to] = now;
43              if (matchr[to] == 0 || dfs(matchr[to])) {
44                  matchr[to] = u; matchl[u] = to;
45                  return 1;
46              }
47          }
48      }
49      return 0;
50  }
51  inline int HK() {
52      int ret = 0;
53      while(bfs()) {
54          ++now;
55          for(int i = 1; i <= Nl; ++i)
56              if(!matchl[i]) ret += dfs(i);
57      }
58      return ret;
59  }
60  void clr() {
61      for(int i = 1; i <= Nl; ++i) {
62          g[i].clear();
63          matchl[i] = 0;
64          levell[i] = 0;
65      }
66      now = 0;
67      for(int i = 1; i <= Nr; ++i) {
68          matchr[i] = 0;
69          levelr[i] = 0;
70          visited[i] = 0;
71      }
72  }
```

## 7.2 最大流

### 7.2.1 最大流（Dinic 算法）

```
1  const int maxn = 1e5 + 5;
2  const int maxm = 2e5 + 5;
```

```
3  const int inf = 0x3f3f3f3f;
4  struct Edge {
5      int to, nxt;
6      int w;
7  } E[maxm << 1];
8  int head[maxn], tot;
9  int n, m, s, t; // vertex number, edge number, source id, sink id
10 void init_edge() {
11     // 1 ~ n
12     for(int i = 1; i <= n; ++i) head[i] = -1;
13     tot = 0;
14 }
15 // undirected edge: add_edge(u, v, w); add_edge(u, v, w);
16 // directed edge: add_edge(u, v, w); add_edge(u, v, 0);
17 void add_edge(int u, int v, int w) {
18     E[tot] = {v, head[u], w};
19     head[u] = tot++;
20 }
21 int dep[maxn];
22 bool bfs() {
23     if(s == t) return true;
24     for(int i = 1; i <= n; ++i) dep[i] = 0;
25     queue<int> q;
26     dep[s] = 1; q.push(s);
27     while(!q.empty()) {
28         int x = q.front(); q.pop();
29         for(int i = head[x]; i != -1; i = E[i].nxt) {
30             int to = E[i].to; int w = E[i].w;
31             if(w == 0 || dep[to] != 0) continue;
32             dep[to] = dep[x] + 1;
33             q.push(to);
34             if(to == t) return true;
35         }
36     }
37     return false;
38 }
39 int cur[maxn];
40 int dfs(int x, int flow) {
41     if(x == t) return flow;
```

```
42      int ret = 0;
43      for(int &i = cur[x]; i != -1; i = E[i].nxt) {
44          int to = E[i].to; int w = E[i].w;
45          if(w == 0 || dep[to] != dep[x] + 1) continue;
46          int now = dfs(to, min(flow - ret, w));
47          if(now > 0) {
48              E[i].w -= now;
49              E[i ^ 1].w += now;
50              ret += now;
51              if(ret == flow) break;
52          }
53      }
54      return ret;
55  }
56  int dinic() {
57      int ret = 0;
58      while(bfs()) {
59          for(int i = 1; i <= n; ++i) cur[i] = head[i];
60          ret += dfs(s, inf);
61      }
62      return ret;
63  }
```

### 7.2.2 最小费用最大流 (dinic + SPFA)

```
1   const int maxn = 1e5 + 5;
2   const int maxm = 2e5 + 5;
3   const int inf = 0x3f3f3f3f;
4   struct Edge {
5       int to, nxt;
6       int cost, flow;
7   } E[maxm << 1];
8   int head[maxn], tot;
9   int n, m, s, t; // point, edge, source, sink
10  int min_cost, max_flow;
11  void init_edge() {
12      for(int i = 1; i <= n; ++i) { // 1 ~ n
13          head[i] = -1;
14      }
```

```
15      tot = 0;
16  }
17  void add_edge(int u, int v, int cost, int flow) {
18      E[tot] = {v, head[u], cost, flow};
19      head[u] = tot++;
20  }
21  void add_network_edge(int u, int v, int cost, int flow) {
22      add_edge(u, v, cost, flow);
23      add_edge(v, u, -cost, 0);
24  }
25  int dist[maxn];
26  bool inq[maxn];
27  bool spfa() {
28      for(int i = 1; i <= n; ++i) {
29          dist[i] = inf;
30          inq[i] = false;
31      }
32      queue<int> q;
33      q.push(s);
34      dist[s] = 0; inq[s] = true;
35      while(!q.empty()) {
36          int x = q.front(); q.pop();
37          for(int i = head[x]; i != -1; i = E[i].nxt) {
38              int to = E[i].to, d = E[i].cost + dist[x], f = E[i].flow;
39              if(f == 0 || d >= dist[to]) continue;
40              dist[to] = d;
41              if(!inq[to]) {
42                  inq[to] = true;
43                  q.push(to);
44              }
45          }
46          inq[x] = false;
47      }
48      if(dist[t] == inf) return false;
49      return true;
50  }
51  int cur[maxn];
52  bool vis[maxn];
53  int dfs(int x, int flow) {
```

```
54      if(x == t) {
55          max_flow += flow;
56          return flow;
57      }
58      vis[x] = true;
59      int ret = 0;
60      for(int &i = cur[x]; i != -1; i = E[i].nxt) {
61          int to = E[i].to, d = dist[x] + E[i].cost, f = E[i].flow;
62          if(vis[to] || f == 0 || d != dist[to]) continue;
63          int now = dfs(to, min(flow - ret, f));
64          if(now > 0) {
65              E[i].flow -= now;
66              E[i ^ 1].flow += now;
67              min_cost += E[i].cost*now;
68              ret += now;
69              if(ret == flow) break;
70          }
71      }
72      return ret;
73  }
74  void min_cost_max_flow() {
75      min_cost = 0; max_flow = 0;
76      while(spfa()) {
77          for(int i = 1; i <= n; ++i) {
78              vis[i] = false;
79              cur[i] = head[i];
80          }
81          dfs(s, inf);
82      }
83  }
```

## 7.3 一般图最大匹配（带花树开花算法）

第一行一个整数，表示最多产生多少个小组。

接下来一行 n 个整数，描述一组最优方案。第 v 个整数表示 v 号男生所在小组的另一个男生的编号。如果 v 号男生没有小组请输出 0。

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  const int maxn = 5e2 + 5;
```

```
4  const int maxm = maxn*(maxn - 1)/2;
5  struct Edge {
6      int to, nxt;
7  } E[maxm << 1];
8  int head[maxn], tot;
9  void init_edge(int n) {
10     for(int i = 0; i <= n; ++i) head[i] = 0;
11     tot = 0;
12 }
13 void add_edge(int x, int y) {
14     E[++tot] = {y, head[x]};
15     head[x] = tot;
16 }
17 void add(int x, int y) {
18     add_edge(x, y);
19     add_edge(y, x);
20 }
21 int *Top;
22 int Stp; // time stamp
23 int n; // vertex number: [1, n]
24 // Nxt[]: match id
25 int Pre[maxn], Nxt[maxn], S[maxn], Q[maxn], Vis[maxn];
26 int fa[maxn];
27 void init_match(int n) {
28     init_edge(n);
29     Top = Q;
30     Stp = 0;
31     for(int i = 0; i <= n; ++i) {
32         fa[i] = i;
33         Pre[i] = Nxt[i] = Vis[i] = 0;
34     }
35 }
36 int findfa(int x) {
37     if(fa[x] != x) fa[x] = findfa(fa[x]);
38     return fa[x];
39 }
40
41 int lca(int x, int y) {
42     for(Stp++, x = findfa(x), y = findfa(y); ; x ^= y ^= x ^= y) {
```

```
43          if(x) {
44              if(Vis[x] == Stp) return x;
45              Vis[x] = Stp;
46              x = findfa(Pre[Nxt[x]]);
47          }
48      }
49  }
50
51  void blossom(int x, int y, int l) {
52      while(findfa(x) != l) {
53          Pre[x] = y, y = Nxt[x];
54          assert(x != -1);
55          S[*Top = y] = 0, *Top++;
56          if(fa[x] == x) fa[x] = l;
57          if(fa[y] == y) fa[y] = l;
58          x = Pre[y];
59      }
60  }
61
62  int match(int x) {
63      for(int i = 1; i <= n; ++i) fa[i] = i;
64      memset(S, -1, sizeof(S));
65      S[*(Top = Q) = x] = 0, Top++;
66      for(int *i = Q; i != Top; *i++)
67          for(int T = head[*i]; T != 0; T = E[T].nxt) {
68              int g = E[T].to;
69              if(S[g] == -1) {
70                  Pre[g] = *i, S[g] = 1;
71                  if(Nxt[g] == 0) {
72                      for(int u = g, v = *i, lst; v; u = lst, v = Pre[u])
73                          lst = Nxt[v], Nxt[v] = u, Nxt[u] = v;
74                      return 1;
75                  }
76                  S[*Top = Nxt[g]] = 0, Top++;
77              } else if(!S[g] && findfa(g) != findfa(*i)) {
78                  int l = lca(g, *i);
79                  blossom(g, *i, l);
80                  blossom(*i, g, l);
81              }
```

```
82          }
83      return 0;
84  }
85  int max_match() {
86      int ret = 0;
87      for(int i = n; i >= 1; --i) {
88          if(!Nxt[i]) ret += match(i);
89      }
90      return ret;
91  }
92  int main() {
93      int m;
94      scanf("%d%d", &n, &m);
95      init_match(n);
96      for(int i = 1; i <= m; ++i) {
97          int x, y; scanf("%d%d", &x, &y);
98          add(x, y);
99      }
100     int ans = max_match();
101     printf("%d\n", ans);
102     for(int i = 1; i <= n; i++) {
103         if(i > 1) printf(" ");
104         printf("%d", Nxt[i]);
105     }
106     puts("");
107     return 0;
108 }
```

## 7.4 平面图相关

### 7.4.1 平面图的性质

平面图的欧拉公式:

如果一个连通的平面图有 n 个点，m 条边和 f 个面，那么 f = m - n + 2

### 7.4.2 平面图 G 与其对偶图 G*

G* 中的每个点对应 G 中的一个面

对于 G 中的每条边 e e 属于两个面 f1、f2，加入边 (f1*, f2*) e 只属于一个面 f，加入回边 (f*, f*)

G 的面数等于 G* 的点数，G* 的点数等于 G 的面数，G 与 G* 边数相同

G* 中的环与 G 中的割一一对应

#### 7.4.3 s-t 平面图

原图为平面图

图中的一个点为源点 s，另外一个点为汇点 t，且 s 和 t 都在图中的无界面的边界上

#### 7.4.4 应用 s-t 平面图求最小割

在平面图中的起点和终点之间加一条边

将平面图（即原来的图）转化为对偶图，记起点和终点构成的边组成的面为源点，无界面为汇点，每条边的权等于原图中对应边的权

删去源点和汇点之间的边

在对偶图上求最短路即为原图的最大流（或最小割）

# 8 数据结构

## 8.1 线段树

### 8.1.1 zkw 线段树

```
template<typename T> struct Segtree {
    static const int N = 1e5 + 5;
    T dat[N << 2]; int siz;

    void init(int n) {
        siz = 1 << int(ceil(log2(n)));
        for (int i = siz; i < siz+n; i++) dat[i] = T();
        for (int i = siz-1; i; i--) dat[i].upd(dat[i << 1], dat[i << 1 |
            1]);
    }
    // 单点查询-区间修改
    T query(int n) {
        T val; int p = siz + n;
        do {
            dat[p].stat(val);
        } while (p >>= 1);
        return val;
    }

    void change(int l, int r, T dt) {
        l += siz, r += siz;
        while(l <= r) {
            if (l & 1) dat[l++].upd(dt);
```

```
            if (!(r & 1)) dat[r--].upd(dt);
            l >>= 1, r >>= 1;
        }
    }

    // 单点修改-区间查询
    void change(int n, int dt) {
        int p = siz + n;
        dat[p].change(dt);
        while (p >>= 1) {
            dat[p].upd(dat[p << 1], dat[p << 1 | 1]);
        }
    }

    T query(int l, int r) {
        T val;
        l += siz, r += siz;
        while(l <= r) {
            if (l & 1) dat[l++].stat(val);
            if (!(r & 1)) dat[r--].stat(val);
            l >>= 1, r >>= 1;
        }
        return val;
    }
};
```

### 8.1.2 动态开点线段树

```
struct SegTree {
    int l, r;
    int ls, rs;
    int sum, lazy;
} st[maxn];
int stc = 0; // Segment Tree counting number
int root; // root of Segment Tree
void pushUp(int x) {
    SegTree& now = st[x];
    SegTree& ls = st[now.ls];
    SegTree& rs = st[now.rs];
```

```
12      now.sum = ls.sum + rs.sum;
13      now.l = ls.l;
14      now.r = rs.r;
15  }
16  void build(int lf, int rt, int x) {
17      SegTree& now = st[x];
18      if(lf == rt) {
19          now.ls = now.rs = now.lazy = -1;
20          now.l = now.r = lf;
21          return ;
22      }
23      int mid = (lf + rt)>>1;
24      now.ls = stc++; now.rs = stc++;
25      build(lf, mid, now.ls);
26      build(mid + 1, rt, now.rs);
27      pushUp(x);
28  }
29  void pushDown(int x) {
30      SegTree& now = st[x];
31      SegTree& ls = st[now.ls];
32      SegTree& rs = st[now.rs];
33      ls.sum = now.lazy*(ls.r - ls.l + 1);
34      rs.sum = now.lazy*(rs.r - rs.l + 1);
35      ls.lazy = now.lazy;
36      rs.lazy = now.lazy;
37      now.lazy = -1;
38  }
39  void update(int lf, int rt, int c, int x) {
40      SegTree& now = st[x];
41      if(lf <= now.l && now.r <= rt) {
42          now.sum = c*(now.r - now.l + 1);
43          now.lazy = c;
44          return ;
45      }
46      if(now.lazy != -1) pushDown(x);
47      int mid = (now.l + now.r)>>1;
48      if(lf <= mid) update(lf, rt, c, now.ls);
49      if(rt > mid) update(lf, rt, c, now.rs);
50      pushUp(x);
```

```
}
int query(int lf, int rt, int x) {
    SegTree& now = st[x];
    if(lf <= now.l && now.r <= rt) return now.sum;
    if(now.lazy != -1) pushDown(x);
    int ret = 0, mid = (now.l + now.r)>>1;
    if(lf <= mid) ret += query(lf, rt, now.ls);
    if(rt > mid) ret += query(lf, rt, now.rs);
    return ret;
}

int main() {
    root = stc++;
    build(1, n, root);

    return 0;
}
```

### 8.1.3 区间最大子段和

有 n 个数，a[1] 到 a[n]。

接下来 q 次查询，每次动态指定两个数 l, r，求 a[l] 到 a[r] 的最大子段和。

子段的意思是连续非空区间。

```
const int maxn = 5e4 + 5;
ll a[maxn];
struct node{
    int lf, rt;
    ll sum;
    ll mxl, mxr, mxa;
} tree[maxn << 2];
void pushUp(int x) {
    node& ls = tree[x<<1];
    node& rs = tree[x<<1|1];
    node& now = tree[x];
    now.sum = ls.sum + rs.sum;
    now.mxl = max(ls.mxl, ls.sum + rs.mxl);
    now.mxr = max(rs.mxr, rs.sum + ls.mxr);
    ll a = ls.mxa, b = rs.mxa, c = ls.mxr + rs.mxl;
    now.mxa = max(a, max(b, c));
```

```
17 }
18 void build(int x, int l, int r){
19     node& now = tree[x];
20     now.lf = l, now.rt = r;
21     if(l == r){
22         now.sum = now.mxa = a[l];
23         now.mxl = now.mxr = a[l];
24         return ;
25     }
26     int mid = (l + r)>>1;
27     int ls = (x<<1), rs = (x<<1)|1;
28     build(ls, l, mid);
29     build(rs, mid + 1, r);
30     pushUp(x);
31 }
32 ll querySum(int x, int l, int r){
33      node& now = tree[x];
34     if(l == now.lf && r == now.rt) return now.sum;
35     int mid = (now.lf + now.rt)>>1;
36     int ls = x<<1, rs = (x<<1)|1;
37     if(mid >= r) return querySum(ls, l, r);
38     if(mid < l) return querySum(rs, l, r);
39     ll ret = querySum(ls, l, mid) + querySum(rs, mid + 1, r);
40     return ret;
41 }
42 ll queryLf(int x, int l, int r){
43     node& now = tree[x];
44     if(l == now.lf && r == now.rt) return now.mxl;
45     int mid = (now.lf + now.rt)>>1;
46     int ls = x<<1, rs = (x<<1)|1;
47     if(mid >= r) return queryLf(ls, l, r);
48     if(mid < l) return queryLf(rs, l, r);
49     ll ret = max(queryLf(ls, l, mid), queryLf(rs, mid + 1, r) + querySum(
           ls, l, mid));
50     return ret;
51 }
52 ll queryRt(int x, int l, int r){
53     node& now = tree[x];
54     if(l == now.lf && r == now.rt) return now.mxr;
```

```
55      int mid = (now.lf + now.rt)>>1;
56      int ls = x<<1, rs = (x<<1)|1;
57      if(mid >= r) return queryRt(ls, l, r);
58      if(mid < l) return queryRt(rs, l, r);
59      ll ret = max(queryRt(rs, mid + 1, r), queryRt(ls, l, mid) + querySum(
            rs, mid + 1, r));
60      return ret;
61  }
62  ll queryAll(int x, int l, int r){
63      node& now = tree[x];
64      if(l == now.lf && r == now.rt) return now.mxa;
65      int mid = (now.lf + now.rt)>>1;
66      int ls = x<<1, rs = (x<<1)|1;
67      if(mid >= r) return queryAll(ls, l, r);
68      if(mid < l) return queryAll(rs, l, r);
69      ll ret = max(queryAll(ls, l, mid), queryAll(rs, mid + 1, r));
70      ret = max(ret, queryLf(rs, mid + 1, r) + queryRt(ls, l, mid));
71      return ret;
72  }
73  int main(){
74      int n; scanf("%d", &n);
75      for(int i = 1; i <= n; ++i) scanf("%lld", &a[i]);
76      build(1, 1, n);
77      int q; scanf("%d", &q);
78      while(q--){
79          int x, y; scanf("%d%d", &x, &y);
80          ll ans = queryAll(1, x, y);
81          printf("%lld\n", ans);
82      }
83      return 0;
84  }
```

### 8.1.4 权值线段树启发式合并

```
1   const int maxn = 1e5 + 5;
2   struct SegTree {
3       int l, r, lc, rc;
4       int num;
5       SegTree() {
```

```
6           l = r = 0;
7           lc = rc = 0;
8           num = 0;
9       }
10  }tree[maxn << 5];  // size = n*log_n
11  int root[maxn], fa[maxn], tot = 0;
12  int reuse[maxn << 5], cnt = 0;
13  void ist(int& x, int lf, int rt, int y) {
14      if(x == 0) {
15          if(cnt == 0) x = ++tot;
16          else x = reuse[--cnt];
17      }
18      SegTree& st = tree[x];
19      st.l = lf, st.r = rt;
20      if(lf == rt) {
21          st.num = 1;
22          return ;
23      }
24      int mid = (lf + rt) >> 1;
25      if(y <= mid) ist(st.lc, lf, mid, y);
26      else ist(st.rc, mid + 1, rt, y);
27      st.num = tree[st.lc].num + tree[st.rc].num;
28  }
29  int kth(int x, int k) {
30      SegTree& st = tree[x];
31      if(k > st.num) return -1;
32      if(st.l == st.r) return st.l;
33      int lc = st.lc, rc = st.rc;
34      if(tree[lc].num >= k) return kth(lc, k);
35      else return kth(rc, k - tree[lc].num);
36  }
37  int query(int x, int lf, int rt) {
38      if(x == 0) return 0;
39      SegTree& st = tree[x];
40      if(st.l >= lf && st.r <= rt) return st.num;
41      int ret = 0;
42      int mid = (st.l + st.r) >> 1;
43      if(mid >= lf) ret += query(st.lc, lf, rt);
44      if(mid < rt) ret += query(st.rc, lf, rt);
```

```
45      return ret;
46  }
47  void del(int x) {
48      if(x == 0) return ;
49      tree[x].l = tree[x].r = 0;
50      tree[x].lc = tree[x].rc = 0;
51      tree[x].num = 0;
52      reuse[cnt++] = x;
53  }
54  int mrg(int x, int y) {
55      if(x == 0) return y;
56      if(y == 0) return x;
57      if(tree[x].num < tree[y].num) swap(x, y);
58      int xlc = tree[x].lc, xrc = tree[x].rc;
59      int ylc = tree[y].lc, yrc = tree[y].rc;
60      tree[x].lc = mrg(xlc, ylc);
61      tree[x].rc = mrg(xrc, yrc);
62      tree[x].num += tree[y].num;
63      del(y);
64      return x;
65  }
```

### 8.1.5 矩形周长并

```
1  #include <cstdio>
2  #include <algorithm>
3  #include <vector>
4  #include <unordered_map>
5  using namespace std;
6  typedef long long ll;
7  const ll mod = 1e9 + 7;
8  const int inf = 0x3f3f3f3f;
9  const ll linf = 0x3f3f3f3f3f3f3f3f;
10 // (max N) * 4
11 const int maxn = (5e4 + 5)*4;
12 struct Rect {
13     int lf, rt, up, down;
14 } r[maxn];
15 struct Event {
```

```
16      int lf, rt, tp;
17      bool operator <(const Event& rhs) const {
18          return tp > rhs.tp;
19      }
20  };
21  vector<Event> E[maxn];
22  struct SegTree {
23      int lf, rt;
24      ll mi, mx, lazy, len;
25      void pt() {
26          printf("[%d, %d]: mi: %lld mx: %lld lazy: %lld len: %lld\n", lf,
                rt, mi, mx, lazy, len);
27      }
28  } st[maxn << 2];
29  vector<int> v;
30  ll len[maxn];
31  void Build(int x, int lf, int rt) {
32      SegTree& now = st[x];
33      now.lf = lf, now.rt = rt;
34      now.mi = 0; now.mx = 0;
35      now.lazy = 0;
36      if(lf == rt) {
37          now.len = len[lf];
38          return ;
39      }
40      int mid = (lf + rt) >> 1;
41      Build(x << 1, lf, mid);
42      Build(x << 1 | 1, mid + 1, rt);
43      now.len = st[x << 1].len + st[x << 1 | 1].len;
44  }
45  void PushUp(int x) {
46      SegTree& now = st[x];
47      const SegTree& lc = st[x << 1];
48      const SegTree& rc = st[x << 1 | 1];
49      now.mi = min(lc.mi, rc.mi);
50      now.mx = max(lc.mx, rc.mx);
51  }
52  void PushDown(int x) {
53      SegTree& now = st[x];
```

```
54      if(now.lazy == 0) return ;
55      SegTree& lc = st[x << 1];
56      SegTree& rc = st[x << 1 | 1];
57      lc.lazy += now.lazy;
58      lc.mi += now.lazy;
59      lc.mx += now.lazy;
60      rc.lazy += now.lazy;
61      rc.mi += now.lazy;
62      rc.mx += now.lazy;
63      now.lazy = 0;
64  }
65  ll add(int x, int lf, int rt) {
66      if(lf > rt) return 0LL;
67      SegTree& now = st[x];
68      if(lf <= now.lf && rt >= now.rt) {
69          if(now.mx <= 0) {
70              ++now.mi; ++now.mx;
71              ++now.lazy;
72              return now.len;
73          }
74          if(now.mi > 0) {
75              ++now.mi; ++now.mx;
76              ++now.lazy;
77              return 0;
78          }
79      }
80      PushDown(x);
81      ll ret = 0;
82      int mid = (now.lf + now.rt) >> 1;
83      if(mid >= lf) ret += add(x << 1, lf, rt);
84      if(mid < rt) ret += add(x << 1 | 1, lf, rt);
85      PushUp(x);
86      return ret;
87  }
88  ll del(int x, int lf, int rt) {
89      if(lf > rt) return 0LL;
90      SegTree& now = st[x];
91      if(lf <= now.lf && rt >= now.rt) {
92          if(now.mx <= 1) {
```

```
93              --now.mi; --now.mx;
94              --now.lazy;
95              return now.len;
96          }
97          if(now.mi > 1) {
98              --now.mi; --now.mx;
99              --now.lazy;
100             return 0;
101         }
102     }
103     PushDown(x);
104     ll ret = 0;
105     int mid = (now.lf + now.rt) >> 1;
106     if(mid >= lf) ret += del(x << 1, lf, rt);
107     if(mid < rt) ret += del(x << 1 | 1, lf, rt);
108     PushUp(x);
109     return ret;
110 }
111 int n, sz;
112 ll solve() {
113     ll ret = 0;
114     Build(1, 1, sz);
115     for(int i = 1; i <= sz; ++i) {
116         sort(E[i].begin(), E[i].end());
117         int sz = E[i].size();
118         for(int j = 0; j < sz; ++j) {
119             Event& now = E[i][j];
120             if(now.tp > 0) ret += add(1, now.lf, now.rt);
121             else ret += del(1, now.lf, now.rt);
122         }
123         E[i].clear();
124     }
125     return ret;
126 }
127 unordered_map<int, int> id;
128 void init() {
129     sort(v.begin(), v.end());
130     v.erase(unique(v.begin(), v.end()), v.end());
131     sz = v.size();
```

```
132     for(int i = 0; i < sz; ++i) {
133         id[v[i]] = i + 1;
134         if(i > 0) len[i] = v[i] - v[i - 1];
135     }
136 }
137 int main(){
138     scanf("%d", &n);
139     for(int i = 0; i < n; ++i) {
140         scanf("%d%d%d%d", &r[i].lf, &r[i].down, &r[i].rt, &r[i].up);
141         v.push_back(r[i].lf); v.push_back(r[i].rt);
142         v.push_back(r[i].down); v.push_back(r[i].up);
143     }
144     init();
145     for(int i = 0; i < n; ++i) {
146         r[i].lf = id[r[i].lf];
147         r[i].rt = id[r[i].rt];
148         r[i].down = id[r[i].down];
149         r[i].up = id[r[i].up];
150     }
151     for(int i = 0; i < n; ++i) {
152         E[r[i].down].push_back({r[i].lf, r[i].rt - 1, 1});
153         E[r[i].up].push_back({r[i].lf, r[i].rt - 1, -1});
154     }
155     ll ans = solve();
156     for(int i = 0; i < n; ++i) {
157         E[r[i].lf].push_back({r[i].down, r[i].up - 1, 1});
158         E[r[i].rt].push_back({r[i].down, r[i].up - 1, -1});
159     }
160     ans += solve();
161     printf("%lld\n", ans);
162     return 0;
163 }
```

### 8.1.6 主席树

区间第 k 大

```
1 const int maxn = 1e5 + 5;
2 struct node{
3     int l, r;
```

```
4       int sum;
5   } T[maxn << 5];
6   int a[maxn], cnt, root[maxn];
7   void update(int l, int r, int& x, int y, int pos){
8       T[++cnt] = T[y]; ++T[cnt].sum; x = cnt;
9       if(l == r) return ;
10      int mid = (l + r) >> 1;
11      if(mid >= pos) update(l, mid, T[x].l, T[y].l, pos);
12      else update(mid + 1, r, T[x].r, T[y].r, pos);
13  }
14  int query(int l, int r, int x, int y, int k){
15      if(l == r) return l;
16      int mid = (l + r) >> 1;
17      int sum = T[T[y].l].sum - T[T[x].l].sum;
18      if(sum >= k) return query(l, mid, T[x].l, T[y].l, k);
19      return query(mid + 1, r, T[x].r, T[y].r, k - sum);
20  }
```

## 8.2 Trie 树（字典树）

```
1   int trie[400001][26];
2   int tot, sum[400001];
3   char s[11];
4   void ist(){
5       int ls = strlen(s);
6       int root = 0;
7       for(int i = 0; i < ls; ++i){
8           int id = s[i] - 'a';
9           if(!trie[root][id]){
10              ++tot;
11              trie[root][id] = tot;
12          }
13          ++sum[trie[root][id]];
14          root = trie[root][id];
15      }
16      return ;
17  }
18
19  int src(){
```

```
    int root = 0;
    int ls = strlen(s);
    for(int i = 0; i < ls; ++i){
        int id = s[i] - 'a';
        if(!trie[root][id])
            return 0;
        root = trie[root][id];
    }
    return sum[root];
}
```

## 8.3 RMQ 问题

```
int n, logn, bs2[maxn], pos[maxn][20];
ll a[maxn], mx[maxn][20];
void RMQ_Init() {
    logn = int(ceil(log2(n)) + eps);
    int bs = 1, num = 1;
    for(int i = 1; i <= n; ++i) {
        mx[i][0] = a[i];
        pos[i][0] = i;
        if((num << 1) <= i) {
            ++bs;
            num <<= 1;
        }
        bs2[i] = bs - 1;
    }
    for(int i = 1; i <= logn; ++i) {
        for(int j = 1; j + (1 << i) - 1 <= n; ++j) {
            if(mx[j][i - 1] > mx[j + (1 << (i - 1))][i - 1]) {
                mx[j][i] = mx[j][i - 1];
                pos[j][i] = pos[j][i - 1];
            } else {
                mx[j][i] = mx[j + (1 << (i - 1))][i - 1];
                pos[j][i] = pos[j + (1 << (i - 1))][i - 1];
            }
        }
    }
}
```

```
ll Qmx; int Qpos;
void Query(int lf, int rt) {
    int bs = bs2[rt - lf + 1];
    if(mx[lf][bs] > mx[rt - (1 << bs) + 1][bs]) {
        Qmx = mx[lf][bs];
        Qpos = pos[lf][bs];
    } else {
        Qmx = mx[rt - (1 << bs) + 1][bs];
        Qpos = pos[rt - (1 << bs) + 1][bs];
    }
}
```

## 8.4 分块

### 8.4.1 分块算法

```
ll a[maxn], sum[maxn], add[maxn];
int l[maxn], r[maxn]; // 每一段的左右端点
int pos[maxn]; // 每一个位置属于哪一段
int n, m; // n 数组大小 m 操作次数

void init(){
    int t = sqrt(n);
    for(int i = 1; i <= t; ++i){
        l[i] = (i - 1)*t + 1;
        r[i] = i*t;
    }
    if(r[t] < n){
        ++t;
        l[t] = r[t - 1] + 1;
        r[t] = n;
    }
    for(int i = 1; i <= t; ++i){
        sum[i] = 0;
        for(int j = l[i]; j <= r[i]; ++j){
            pos[j] = i;
            sum[i] += a[j];
        }
    }
}
```

```
25 void update(int lf, int rt, ll d){ // [lf, rt] 区间内的数全部加 d
26     int p = pos[lf], q = pos[rt];
27     if(p == q){
28         for(int i = lf; i <= rt; ++i){
29             a[i] += d;
30             sum[p] += d;
31         }
32         return ;
33     }
34     for(int i = lf; i <= r[p]; ++i){
35         a[i] += d;
36         sum[p] += d;
37     }
38     for(int i = p + 1; i <= q - 1; ++i) add[i] += d;
39     for(int i = l[q]; i <= rt; ++i){
40         a[i] += d;
41         sum[q] += d;
42     }
43 }
44
45 ll query(int lf, int rt){
46     ll ret = 0;
47     int p = pos[lf], q = pos[rt];
48     if(p == q){
49         for(int i = lf; i <= rt; ++i) ret += a[i] + add[p];
50         return ret;
51     }
52     for(int i = lf; i <= r[p]; ++i) ret += a[i] + add[p];
53     for(int i = p + 1; i <= q - 1; ++i) ret += sum[i] + add[i]*(r[i] - l[i
       ] + 1LL);
54     for(int i = l[q]; i <= rt; ++i) ret += a[i] + add[q];
55     return ret;
56 }
```

### 8.4.2 莫队算法

```
1 const int maxn = 2e6 + 5;
2 int L[maxn], R[maxn], pos[maxn], cnt[maxn];
3 ll num[maxn], ans[maxn], A = 0;
```

```
4  int n, m;
5  struct Query {
6      int l, r, id;
7      bool operator < (const Query& q) const {
8          if(pos[l] == pos[q.l]) {
9              if(pos[l] & 1) return r > q.r;
10             else return r < q.r;
11         }
12         return pos[l] < pos[q.l];
13     }
14 }Q[maxn];
15
16 void init(){
17     int t = sqrt(n);
18     for(int i = 1; i <= t; ++i){
19         L[i] = (i - 1)*t + 1;
20         R[i] = i*t;
21     }
22     if(R[t] < n){
23         ++t;
24         L[t] = R[t - 1] + 1;
25         R[t] = n;
26     }
27     for(int i = 1; i <= t; ++i) for(int j = L[i]; j <= R[i]; ++j) pos[j] =
           i;
28 }
29
30 void del(int x){
31     if(--cnt[num[x]] == 0) A -= num[x];
32 }
33
34 void add(int x){
35     if(++cnt[num[x]] == 1) A += num[x];
36 }
37 void Mo() {
38     init();
39     sort(Q, Q + m);
40     int lf = 1, rt = 0;
41     A = 0;
```

```
    for(int i = 0; i < m; ++i){
        while(lf < Q[i].l) del(lf++);
        while(lf > Q[i].l) add(--lf);
        while(rt < Q[i].r) add(++rt);
        while(rt > Q[i].r) del(rt--);
        ans[Q[i].id] = A;
    }
}
int main(){
    int t; scanf("%d", &t);
    while(t--) {
        scanf("%d", &n);
        ll mx = 0;
        for(int i = 1; i <= n; ++i){
            scanf("%lld", &num[i]);
            if(num[i] > mx) mx = num[i];
        }
        for(int i = 0; i <= mx; ++i) cnt[i] = 0;
        scanf("%d", &m);
        for(int i = 0; i < m; ++i){
            scanf("%d%d", &Q[i].l, &Q[i].r);
            Q[i].id = i;
        }
        Mo();
        for(int i = 0; i < m; ++i) printf("%lld\n", ans[i]);
    }
    return 0;
}
```

### 8.4.3 待修改莫队

块大小为 $(nt)^{\frac{1}{3}}$ 可以达到最快的理论复杂度 $O((n^4t)^{\frac{1}{3}})$ $n$ 为数组大小，$t$ 为修改次数

墨墨购买了一套 N 支彩色画笔（其中有些颜色可能相同），摆成一排，你需要回答墨墨的提问。墨墨会像你发布如下指令：

Q L R 代表询问你从第 L 支画笔到第 R 支画笔中共有几种不同颜色的画笔。

R P Col 把第 P 支画笔替换为颜色 Col。

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

```

```
4  const int maxn = 1e6 + 5;
5  int L[maxn], R[maxn], pos[maxn];
6  int a[maxn], b[maxn];
7  int cnt[maxn], ans[maxn], Ans = 0;
8  int n, m, k, nq = 0, nc = 0;
9  struct Query {
10     int l, r;
11     int pre, id;
12     bool operator < (const Query& q) const {
13         if(pos[l] == pos[q.l]) {
14             if(pos[r] == pos[q.r]) return pre < q.pre;
15             return pos[r] < pos[q.r];
16         }
17         return pos[l] < pos[q.l];
18     }
19 } Q[maxn];
20 struct Change {
21     int pos, bfr, aft;
22 } C[maxn];
23
24 void init() {
25     double val = (double)n*(1.0 + nc);
26     int sz = pow(val, 0.333333333);
27     int num = n/sz;
28     for(int i = 1; i <= num; ++i){
29         L[i] = (i - 1)*sz + 1;
30         R[i] = i*sz;
31     }
32     if(R[num] < n){
33         ++num;
34         L[num] = R[num - 1] + 1;
35         R[num] = n;
36     }
37     for(int i = 1; i <= num; ++i) for(int j = L[i]; j <= R[i]; ++j) pos[j]
           = i;
38 }
39 void add(int x) {
40     if(++cnt[a[x]] == 1) ++Ans;
41 }
```

```
42 void del(int x) {
43     if(--cnt[a[x]] == 0) --Ans;
44 }
45 void update(int x, int lf, int rt) {
46     if(C[x].pos >= lf && C[x].pos <= rt) {
47         if(--cnt[C[x].bfr] == 0) --Ans;
48         if(++cnt[C[x].aft] == 1) ++Ans;
49     }
50     a[C[x].pos] = C[x].aft;
51 }
52 void revoke(int x, int lf, int rt) {
53     if(C[x].pos >= lf && C[x].pos <= rt) {
54         if(--cnt[C[x].aft] == 0) --Ans;
55         if(++cnt[C[x].bfr] == 1) ++Ans;
56     }
57     a[C[x].pos] = C[x].bfr;
58 }
59 void Mo() {
60     init(); Ans = 0;
61     sort(Q, Q + nq);
62     int lf = 1, rt = 0, now = 0;
63     for(int i = 0; i < nq; ++i) {
64         while(lf < Q[i].l) del(lf++);
65         while(lf > Q[i].l) add(--lf);
66         while(rt < Q[i].r) add(++rt);
67         while(rt > Q[i].r) del(rt--);
68         while(now < Q[i].pre) update(++now, Q[i].l, Q[i].r);
69         while(now > Q[i].pre) revoke(now--, Q[i].l, Q[i].r);
70         ans[Q[i].id] = Ans;
71     }
72 }
73 char s[5];
74 int main(){
75     scanf("%d%d", &n, &m);
76     for(int i = 1; i <= n; ++i) {
77         scanf("%d", &a[i]);
78         b[i] = a[i];
79     }
80     for(int i = 1; i <= 1000000; ++i) cnt[i] = 0;
```

```
    while(m--) {
        scanf("%s", s);
        if(s[0] == 'Q') {
            scanf("%d%d", &Q[nq].l, &Q[nq].r);
            Q[nq].pre = nc; Q[nq].id = nq;
            ++nq;
        } else {
            ++nc;
            scanf("%d%d", &C[nc].pos, &C[nc].aft);
            C[nc].bfr = b[C[nc].pos];
            b[C[nc].pos] = C[nc].aft;
        }
    }
    Mo();
    for(int i = 0; i < nq; ++i) printf("%d\n", ans[i]);
    return 0;
}
```

## 8.5 树链剖分（轻重链剖分）+ 线段树维护链上信息

Heavy-light Decomposition : HLD

边剖分可以将权赋给一条边上 dep 更大的那个节点，转化成点权问题。

最后 x 和 y 在同一个重链的时候，直接 $update(pos[x]+1, pos[y], 1, maxn, 1)$

注意是 $pos[x]+1$.

### 8.5.1 模板

```
struct Edge {
    int to, nxt;
    int v;
    Edge(int to = 0, int nxt = 0, int v = 0): to(to), nxt(nxt), v(v){}
} e[maxn << 1];
int head[maxn], tot;
void init(){
    memset(head, -1, sizeof(head));
    tot = 0;
}
void addEdge(int x, int y, int z){
    e[tot] = Edge(y, head[x], z);
    head[x] = tot++;
```

```
14 }
15
16 int rk[maxn]; // dfs order to original order
17
18 struct TreeChain {
19     int fa, d, sz;
20     int hs; // heavy son
21     int tp; // top of heavy link
22     int id; // original order to dfs order
23     TreeChain(){
24         hs = 0;
25     }
26 } tc[maxn];
27 int tcc; // Tree Chain dfs order
28 void dfs1(int x, int y, int d){
29     TreeChain& now = tc[x];
30     now.sz = 1; now.fa = y;
31     now.d = d;
32     for(int i = head[x]; i != -1; i = e[i].nxt){
33         int to = e[i].to;
34         if(to == y) continue;
35         dfs1(to, x, d + 1);
36         now.sz += tc[to].sz;
37         int hs = now.hs;
38         if(hs == 0 || tc[hs].sz < tc[to].sz)
39             now.hs = to;
40     }
41 }
42 void dfs2(int x, int y, int z){
43     TreeChain& now = tc[x];
44     now.tp = z; now.id = tcc;
45     rk[tcc++] = x;
46     int hs = now.hs;
47     if(hs == 0) return ;
48     dfs2(hs, x, z);
49     for(int i = head[x]; i != -1; i = e[i].nxt){
50         int to = e[i].to;
51         if(to == y || to == hs) continue;
52         dfs2(to, x, to);
```

```
    }
}
void TreeChainPartition(int root){
    tcc = 1;
    memset(tc, 0, sizeof(tc));
    dfs1(root, -1, 0);
    dfs2(root, -1, root);
}
```

### 8.5.2 树链剖分维护软件依赖关系

给你一棵树，有两种操作：
把一个节点变为 1，把一个节点变为 0
如果一个节点变为 1，那么他的所有祖先节点都变为 1.
如果一个节点变为 0，那么他的所有后代节点都变为 0.
对于每一个操作，输出发生改变的节点数量。

```
const int maxn = 3e5 + 3;
struct Edge{
    int to, nxt;
    Edge(int to = 0, int nxt = 0): to(to), nxt(nxt){}
} e[maxn << 1];
int head[maxn], tot;
void init(){
    memset(head, -1, sizeof(head));
    tot = 0;
}
void addEdge(int x, int y){
    e[tot] = Edge(y, head[x]);
    head[x] = tot++;
}

int rk[maxn]; // dfs order to original order

struct TreeChain{
    int fa, d, sz;
    int hs; // heavy son
    int tp; // top of heavy link
    int id; // original order to dfs order
    TreeChain(){
```

```
24          hs = -1;
25      }
26  } tc[maxn];
27  int tcc; // Tree Chain dfs order
28  void dfs1(int x, int y, int d){
29      TreeChain& now = tc[x];
30      now.sz = 1; now.fa = y;
31      now.d = d;
32      for(int i = head[x]; i != -1; i = e[i].nxt){
33          int to = e[i].to;
34          if(to == y) continue;
35          dfs1(to, x, d + 1);
36          now.sz += tc[to].sz;
37          int hs = now.hs;
38          if(hs == -1 || tc[hs].sz < tc[to].sz)
39              now.hs = to;
40      }
41  }
42  void dfs2(int x, int y, int z){
43      TreeChain& now = tc[x];
44      now.tp = z; now.id = tcc;
45      rk[tcc++] = x;
46      int hs = now.hs;
47      if(hs == -1) return ;
48      dfs2(hs, x, z);
49      for(int i = head[x]; i != -1; i = e[i].nxt){
50          int to = e[i].to;
51          if(to == y || to == hs) continue;
52          dfs2(to, x, to);
53      }
54  }
55  void TreeChainPartition(int root){
56      tcc = 1;
57      dfs1(root, -1, 0);
58      dfs2(root, -1, root);
59  }
60
61  struct SegTree{
62      int l, r;
```

```
63      int ls, rs;
64      int sum, lazy;
65  }st[maxn<<2];
66  int stc = 0; // Segment Tree counting number
67  int root; // root of Segment Tree
68  void pushUp(int x){
69      SegTree& now = st[x];
70      SegTree& ls = st[now.ls];
71      SegTree& rs = st[now.rs];
72      now.sum = ls.sum + rs.sum;
73      now.l = ls.l;
74      now.r = rs.r;
75  }
76  void build(int lf, int rt, int x){
77      SegTree& now = st[x];
78      if(lf == rt){
79          now.ls = now.rs = now.lazy = -1;
80          now.l = now.r = lf;
81          return ;
82      }
83      int mid = (lf + rt)>>1;
84      now.ls = stc++; now.rs = stc++;
85      build(lf, mid, now.ls);
86      build(mid + 1, rt, now.rs);
87      pushUp(x);
88  }
89  void pushDown(int x){
90      SegTree& now = st[x];
91      SegTree& ls = st[now.ls];
92      SegTree& rs = st[now.rs];
93      ls.sum = now.lazy*(ls.r - ls.l + 1);
94      rs.sum = now.lazy*(rs.r - rs.l + 1);
95      ls.lazy = now.lazy;
96      rs.lazy = now.lazy;
97      now.lazy = -1;
98  }
99  void update(int lf, int rt, int c, int x){
100     SegTree& now = st[x];
101     if(lf <= now.l && now.r <= rt){
```

```
102         now.sum = c*(now.r - now.l + 1);
103         now.lazy = c;
104         return ;
105     }
106     if(now.lazy != -1) pushDown(x);
107     int mid = (now.l + now.r)>>1;
108     if(lf <= mid) update(lf, rt, c, now.ls);
109     if(rt > mid) update(lf, rt, c, now.rs);
110     pushUp(x);
111 }
112 int query(int lf, int rt, int x){
113     SegTree& now = st[x];
114     if(lf <= now.l && now.r <= rt) return now.sum;
115     if(now.lazy != -1) pushDown(x);
116     int ret = 0, mid = (now.l + now.r)>>1;
117     if(lf <= mid) ret += query(lf, rt, now.ls);
118     if(rt > mid) ret += query(lf, rt, now.rs);
119     return ret;
120 }
121
122 int install(int x){
123     int ret = 0, y = tc[x].tp;
124     while(y != 0){
125         int lf = tc[y].id, rt = tc[x].id;
126         ret += rt - lf + 1 - query(lf, rt, root);
127         update(lf, rt, 1, root);
128         x = tc[y].fa;
129         y = tc[x].tp;
130     }
131     int lf = tc[y].id, rt = tc[x].id;
132     ret += rt - lf + 1 - query(lf, rt, root);
133     update(lf, rt, 1, root);
134     return ret;
135 }
136
137 int uninstall(int x){
138     int lf = tc[x].id, rt = tc[x].id + tc[x].sz - 1;
139     int ret = query(lf, rt, root);
140     update(lf, rt, 0, root);
```

```
    return ret;
}
char s[15];
int main(){
    init();
    int n; scanf("%d", &n);
    for(int i = 1; i < n; ++i){
        int x; scanf("%d", &x);
        addEdge(x, i);
        addEdge(i, x);
    }
    TreeChainPartition(0);
    root = stc++;
    build(1, n, root);
    int q; scanf("%d", &q);
    while(q--){
        int x; scanf("%s %d", s, &x);
        if(s[0] == 'i') printf("%d\n", install(x));
        else printf("%d\n", uninstall(x));
    }
    return 0;
}
```

### 8.5.3 边权问题

给出一棵树，每条边编号 $[1, n-1]$ 有权值，可修改第 i 条边的权值，支持询问从节点 x 到节点 y 的路径上的最大边权。

```
const int maxn = 1e5 + 5;
int ea[maxn], eb[maxn], ec[maxn];
struct Edge{
    int to, nxt;
    int v;
    Edge(int to = 0, int nxt = 0, int v = 0): to(to), nxt(nxt), v(v){}
}e[maxn<<1];
int head[maxn], tot;
void init(){
    memset(head, -1, sizeof(head));
    tot = 0;
}
```

```
13 void addEdge(int x, int y, int z){
14     e[tot] = Edge(y, head[x], z);
15     head[x] = tot++;
16 }
17
18 int rk[maxn]; // dfs order to original order
19
20 struct TreeChain{
21     int fa, d, sz;
22     int hs; // heavy son
23     int tp; // top of heavy link
24     int id; // original order to dfs order
25     TreeChain(){
26         hs = 0;
27     }
28 }tc[maxn];
29 int tcc; // Tree Chain dfs order
30 void dfs1(int x, int y, int d){
31     TreeChain& now = tc[x];
32     now.sz = 1; now.fa = y;
33     now.d = d;
34     for(int i = head[x]; i != -1; i = e[i].nxt){
35         int to = e[i].to;
36         if(to == y) continue;
37         dfs1(to, x, d + 1);
38         now.sz += tc[to].sz;
39         int hs = now.hs;
40         if(hs == 0 || tc[hs].sz < tc[to].sz)
41             now.hs = to;
42     }
43 }
44 void dfs2(int x, int y, int z){
45     TreeChain& now = tc[x];
46     now.tp = z; now.id = tcc;
47     rk[tcc++] = x;
48     int hs = now.hs;
49     if(hs == 0) return ;
50     dfs2(hs, x, z);
51     for(int i = head[x]; i != -1; i = e[i].nxt){
```

```
52          int to = e[i].to;
53          if(to == y || to == hs) continue;
54          dfs2(to, x, to);
55      }
56  }
57  void TreeChainPartition(int root){
58      tcc = 1;
59      memset(tc, 0, sizeof(tc));
60      dfs1(root, -1, 0);
61      dfs2(root, -1, root);
62  }
63
64  struct SegTree{
65      int l, r;
66      int ls, rs;
67      int mx;
68  }st[maxn<<2];
69  int stc = 0; // Segment Tree counting number
70  int root; // root of Segment Tree
71  void pushUp(int x){
72      SegTree& now = st[x];
73      SegTree& ls = st[now.ls];
74      SegTree& rs = st[now.rs];
75      now.mx = max(ls.mx, rs.mx);
76      now.l = ls.l;
77      now.r = rs.r;
78  }
79  void build(int lf, int rt, int x){
80      SegTree& now = st[x];
81      if(lf == rt){
82          now.ls = now.rs = -1;
83          now.l = now.r = lf;
84          now.mx = -0x3f3f3f3f;
85          return ;
86      }
87      int mid = (lf + rt)>>1;
88      now.ls = stc++; now.rs = stc++;
89      build(lf, mid, now.ls);
90      build(mid + 1, rt, now.rs);
```

```
91      pushUp(x);
92  }
93  void update(int pos, int c, int x){
94      SegTree& now = st[x];
95      if(now.l == now.r){
96          now.mx = c;
97          return ;
98      }
99      int mid = (now.l + now.r)>>1;
100     if(pos <= mid) update(pos, c, now.ls);
101     if(pos > mid) update(pos, c, now.rs);
102     pushUp(x);
103 }
104 int query(int lf, int rt, int x){
105     SegTree& now = st[x];
106     if(lf <= now.l && now.r <= rt) return now.mx;
107     int ret = -0x3f3f3f3f, mid = (now.l + now.r)>>1;
108     if(lf <= mid) ret = max(ret, query(lf, rt, now.ls));
109     if(rt > mid) ret = max(ret, query(lf, rt, now.rs));
110     return ret;
111 }
112
113 int QUERY(int x, int y){
114     int ret = -0x3f3f3f3f;
115     int tx = tc[x].tp, ty = tc[y].tp;
116     while(tx != ty){
117         if(tc[tx].d < tc[ty].d){
118             swap(tx, ty);
119             swap(x, y);
120         }
121         ret = max(ret, query(tc[tx].id, tc[x].id, root));
122         x = tc[tx].fa; tx = tc[x].tp;
123     }
124     if(x == y) return ret;
125     if(tc[x].d > tc[y].d) swap(x, y);
126     ret = max(ret, query(tc[tc[x].hs].id, tc[y].id, root));
127     return ret;
128 }
129 void CHANGE(int x, int y){
```

```
    update(tc[eb[x]].id, y, root);
}
char s[15];

int main(){
    int t; scanf("%d", &t);
    while(t--){
        init();
        int n; scanf("%d", &n);
        for(int i = 1; i < n; ++i){
            int a, b, c; scanf("%d%d%d", &a, &b, &c);
            addEdge(a, b, c);
            addEdge(b, a, c);
            ea[i] = a, eb[i] = b, ec[i] = c;
        }
        TreeChainPartition(1);
        stc = 0;
        root = stc++;
        build(1, n, root);
        for(int i = 1; i < n; ++i){
            if(tc[ea[i]].d > tc[eb[i]].d) swap(ea[i], eb[i]);
            update(tc[eb[i]].id, ec[i], root);
        }
        while(scanf("%s", s) != EOF){
            if(s[0] == 'D') break;
            int x, y; scanf("%d %d", &x, &y);
            if(s[0] == 'Q') printf("%d\n", QUERY(x, y));
            else CHANGE(x, y);
        }
        if(t) puts("");
    }
    return 0;
}
```

### 8.5.4 点权问题

给出一棵树，每个点编号 $[1,n]$ 有权值，可修改第 i 个点的权值，支持询问从节点 x 到节点 y 的路径上的最大点权。

```
const int maxn = 3e4 + 5;
```

```
const ll inf = 0x3f3f3f3f3f3f3f3f;

struct Edge{
    int to, nxt;
    int v;
    Edge(int to = 0, int nxt = 0, int v = 0):to(to), nxt(nxt), v(v){}
}e[maxn<<1];
int head[maxn], tot;
void init(){
    memset(head, -1, sizeof(head));
    tot = 0;
}
void addEdge(int x, int y, int z){
    e[tot] = Edge(y, head[x], z);
    head[x] = tot++;
}

int rk[maxn];

struct TreeChain{
    int fa, d, sz;
    int hs, tp, id;
    TreeChain(){
        hs = 0;
    }
}tc[maxn];
int tcc;
void dfs1(int x, int y, int d){
    TreeChain& now = tc[x];
    now.sz = 1, now.fa = y;
    now.d = d;
    for(int i = head[x]; i != -1; i = e[i].nxt){
        int to = e[i].to;
        if(to == y) continue;
        dfs1(to, x, d + 1);
        now.sz += tc[to].sz;
        int hs = now.hs;
        if(hs == 0 || tc[hs].sz < tc[to].sz) now.hs = to;
    }
```

```
}

void dfs2(int x, int y, int z){
    TreeChain& now = tc[x];
    now.tp = z; now.id = tcc;
    rk[tcc++] = x;
    int hs = now.hs;
    if(hs == 0) return ;
    dfs2(hs, x, z);
    for(int i = head[x]; i != -1; i = e[i].nxt){
        int to = e[i].to;
        if(to == y || to == hs) continue;
        dfs2(to, x, to);
    }
}

void TreeChainPartition(int root){
    tcc = 1;
    memset(tc, 0, sizeof(tc));
    dfs1(root, -1, 0);
    dfs2(root, -1, root);
}

struct SegTree{
    int l, r;
    int ls, rs;
    ll mx, sum;
}st[maxn<<2];

int stc = 0;
int root;

ll a[maxn];

void pushUp(int x){
    SegTree& now = st[x];
    SegTree& ls = st[now.ls];
    SegTree& rs = st[now.rs];
    now.mx = max(ls.mx, rs.mx);
```

```
80      now.sum = ls.sum + rs.sum;
81  }
82
83  void build(int x, int lf, int rt){
84      SegTree& now = st[x];
85      now.l = lf, now.r = rt;
86      if(lf == rt){
87          now.ls = now.rs = -1;
88          now.mx = -inf;
89          now.sum = 0;
90          return ;
91      }
92      int mid = (lf + rt)>>1;
93      now.ls = stc++; now.rs = stc++;
94      build(now.ls, lf, mid);
95      build(now.rs, mid + 1, rt);
96      pushUp(x);
97  }
98
99  void update(int x, int pos, ll val){
100     SegTree& now = st[x];
101     if(now.l == now.r){
102         now.mx = val;
103         now.sum = val;
104         return ;
105     }
106     int mid = (now.l + now.r)>>1;
107     if(pos <= mid) update(now.ls, pos, val);
108     else update(now.rs, pos, val);
109     pushUp(x);
110 }
111
112 ll queryMax(int x, int lf, int rt){
113     SegTree& now = st[x];
114     if(lf <= now.l && now.r <= rt) return now.mx;
115     ll ret = -inf; int mid = (now.l + now.r)>>1;
116     if(lf <= mid) ret = max(ret, queryMax(now.ls, lf, rt));
117     if(rt > mid) ret = max(ret, queryMax(now.rs, lf, rt));
118     return ret;
```

```
}

ll querySum(int x, int lf, int rt){
    SegTree& now = st[x];
    if(lf <= now.l && now.r <= rt) return now.sum;
    ll ret = 0; int mid = (now.l + now.r)>>1;
    if(lf <= mid) ret += querySum(now.ls, lf, rt);
    if(rt > mid) ret += querySum(now.rs, lf, rt);
    return ret;
}
char s[15];

ll QMAX(int x, int y){
    ll ret = -inf;
    int tx = tc[x].tp, ty = tc[y].tp;
    while(tx != ty){
        if(tc[tx].d < tc[ty].d){
            swap(tx, ty);
            swap(x, y);
        }
        ret = max(ret, queryMax(root, tc[tx].id, tc[x].id));
        x = tc[tx].fa; tx = tc[x].tp;
    }
    if(tc[x].d > tc[y].d) swap(x, y);
    ret = max(ret, queryMax(root, tc[x].id, tc[y].id));
    return ret;
}

ll QSUM(int x, int y){
    ll ret = 0;
    int tx = tc[x].tp, ty = tc[y].tp;
    while(tx != ty){
        if(tc[tx].d < tc[ty].d){
            swap(tx, ty);
            swap(x, y);
        }
        ret += querySum(root, tc[tx].id, tc[x].id);
        x = tc[tx].fa; tx = tc[x].tp;
    }
```

```
158     if(tc[x].d > tc[y].d) swap(x, y);
159     ret += querySum(root, tc[x].id, tc[y].id);
160     return ret;
161 }
162
163 void CHANGE(int x, ll y){
164     update(root, tc[x].id, y);
165 }
166
167 int main(){
168     init();
169     int n; scanf("%d", &n);
170     for(int i = 1; i < n; ++i){
171         int u, v;
172         scanf("%d%d", &u, &v);
173         addEdge(u, v, 0);
174         addEdge(v, u, 0);
175     }
176     TreeChainPartition(1);
177     stc = 0;
178     root = stc++;
179     build(root, 1, n);
180     for(int i = 1; i <= n; ++i){
181         int x; scanf("%d", &x);
182         update(root, tc[i].id, x);
183     }
184     int q; scanf("%d", &q);
185     while(q--){
186         scanf("%s", s);
187         int x, y; scanf("%d%d", &x, &y);
188         if(s[1] == 'M'){
189             ll ans = QMAX(x, y);
190             printf("%lld\n", ans);
191         }else if(s[1] == 'S'){
192             ll ans = QSUM(x, y);
193             printf("%lld\n", ans);
194         }else CHANGE(x, (ll)y);
195     }
196     return 0;
```

```
197 }
```

## 8.6 带权并查集

```
1  int fa[maxn], r[maxn];
2  int findfa(int x){
3      if(x != fa[x]){
4          int fx = fa[x];
5          fa[x] = findfa(fx);
6          r[x] = (r[fx] + r[x])%3;
7      }
8      return fa[x];
9  }
10 void mrg(int x, int y, int d){
11     int fx = findfa(x), fy = findfa(y);
12     if(fx == fy) return ;
13     fa[fx] = fy;
14     r[fx] = (d + r[y] - r[x] + 3)%3;
15 }
16
17 int fa[maxn], w[maxn];
18 int findfa(int x) {
19     if(fa[x] != x) {
20         int fx = fa[x];
21         fa[x] = findfa(fx);
22         w[x] += w[fx];
23     }
24     return fa[x];
25 }
26 bool mrg(int x, int y, int z) {
27     int fx = findfa(x);
28     int fy = findfa(y);
29     if(fx != fy) {
30         fa[fy] = fx;
31         w[fy] = z + w[x] - w[y];
32     } else {
33         if(w[x] + z != w[y]) return false;
34     }
35     return true;
```

```
}
```

## 8.7 二维树状数组

### 8.7.1 单点修改 + 区间查询

```
const int maxn = 1e3 + 5;
#define lowbit(x) (x&(-x))
int tree[maxn][maxn];
void update(int x, int y, int z){
    for(int i = x; i < maxn; i += lowbit(i))
        for(int j = y; j < maxn; j += lowbit(j))
            tree[i][j] += z;
}
int query(int x, int y){
    int ret = 0;
    for(int i = x; i > 0; i -= lowbit(i))
        for(int j = y; j > 0; j -= lowbit(j))
            ret += tree[i][j];
    return ret;
}
```

### 8.7.2 区间修改 + 单点查询

二维前缀和:

sum[i][j] = sum[i − 1][j] + sum[i][j − 1] − sum[i − 1][j − 1] + a[i][j]

那么我们可以令差分数组 d[i][j] 表示 a[i][j] 与 a[i − 1][j] + a[i][j − 1] − a[i − 1][j − 1] 的差。

我们在差分数组 d[][] 上建树状数组，便可以维护区间修改的值，并支持单点查询

```
const int maxn = 1e3 + 5;
#define lowbit(x) (x&(-x))
int tree[maxn][maxn];
void update(int x, int y, int z){
    for(int i = x; i < maxn; i += lowbit(i))
        for(int j = y; j < maxn; j += lowbit(j))
            tree[i][j] += z;
}
void rangeUpdate(int xa, int ya, int xb, int yb, int z){
    update(xa, ya, z);
    update(xa, yb + 1, -z);
```

```
12      update(xb + 1, ya, -z);
13      update(xb + 1, yb + 1, z);
14  }
15  int query(int x, int y){
16      int ret = 0;
17      for(int i = x; i > 0; i -= lowbit(i))
18          for(int j = y; j > 0; j -= lowbit(j))
19              ret += tree[i][j];
20      return ret;
21  }
```

### 8.7.3 区间修改 + 区间查询（待补）

## 8.8 离线分治算法

### 8.8.1 CDQ 分治三维偏序（陌上花开）

有 n 朵花，每朵花有三个属性：花形 (s)、颜色 (c)、气味 (m)，由三个整数表示。

现要对每朵花评级，一朵花的级别是它拥有的美丽能超过的花的数量。定义一朵花 A 比另一朵花 B 要美丽，当且仅当 Sa >= Sb, Ca >= Cb, Ma >= Mb。显然，两朵花可能有同样的属性。

需要统计出评出每个等级的花的数量。

```
1   const int maxn = 4e5 + 5;
2   struct node{
3       int s, c, m;
4       int id;
5       int num, ans;
6       bool operator < (const node& nd) const{
7           if(s == nd.s){
8               if(c == nd.c) return m < nd.m;
9               return c < nd.c;
10          }
11          return s < nd.s;
12      }
13      bool operator == (const node& nd) const{
14          if(s != nd.s) return false;
15          if(c != nd.c) return false;
16          if(m != nd.m) return false;
17          return true;
18      }
19  }a[maxn], b[maxn], c[maxn];
```

```
20 int cnt[maxn];
21 #define lowbit(x) (x&(-x))
22 int tree[maxn];
23 void update(int x, int val){
24     while(x < maxn){
25         tree[x] += val;
26         x += lowbit(x);
27     }
28 }
29 int query(int x){
30     int ret = 0;
31     while(x){
32         ret += tree[x];
33         x -= lowbit(x);
34     }
35     return ret;
36 }
37 void cdq(int lf, int rt){
38     if(lf == rt){
39         b[lf].ans = b[lf].num - 1;
40         return ;
41     }
42     int mid = (lf + rt)>>1;
43     cdq(lf, mid); cdq(mid + 1, rt);
44     int p1 = lf, p2 = mid + 1;
45     int p3 = lf;
46     while(p1 <= mid && p2 <= rt){
47         if(b[p1].c <= b[p2].c){
48             update(b[p1].m, b[p1].num);
49             c[p3++] = b[p1++];
50         }else{
51             b[p2].ans += query(b[p2].m);
52             c[p3++] = b[p2++];
53         }
54     }
55     while(p1 <= mid){
56         update(b[p1].m, b[p1].num);
57         c[p3++] = b[p1++];
58     }
```

```
59      while(p2 <= rt){
60          b[p2].ans += query(b[p2].m);
61          c[p3++] = b[p2++];
62      }
63      for(int i = lf; i <= mid; ++i) update(b[i].m, -b[i].num);
64      for(int i = lf; i <= rt; ++i) b[i] = c[i];
65  }
66  int main(){
67      int n, k; scanf("%d%d", &n, &k);
68      for(int i = 1; i <= n; ++i){
69          scanf("%d%d%d", &a[i].s, &a[i].c, &a[i].m);
70          a[i].id = i;
71      }
72      sort(a + 1, a + n + 1);
73      int tot = 0;
74      b[++tot] = a[1];
75      b[tot].ans = 0; b[tot].num = 1;
76      for(int i = 2; i <= n; ++i){
77          if(a[i] == a[i - 1]) ++b[tot].num;
78          else{
79              b[++tot] = a[i];
80              b[tot].ans = 0;
81              b[tot].num = 1;
82          }
83      }
84      cdq(1, tot);
85      for(int i = 1; i <= tot; ++i) cnt[b[i].ans] += b[i].num;
86      for(int i = 0; i < n; ++i) printf("%d\n", cnt[i]);
87      return 0;
88  }
```

### 8.8.2 基于值域的整体分治算法

求解区间第 k 小

```
1  const int maxn = 2e5 + 5;
2  struct node {
3      int op, x, y, z;
4  } a[maxn], b[maxn], c[maxn];
5  int tree[maxn], ans[maxn], n, m;
```

```
6  #define lowbit(x) (x&(-x))
7  void add(int x, int y) {
8      while(x <= n) {
9          tree[x] += y;
10         x += lowbit(x);
11     }
12 }
13 int query(int x) {
14     int ret = 0;
15     while(x) {
16         ret += tree[x];
17         x -= lowbit(x);
18     }
19     return ret;
20 }
21 void solve(int mi, int mx, int lf, int rt) {
22     if(lf > rt) return ;
23     if(mi == mx) {
24         for(int i = lf; i <= rt; ++i) ans[a[i].op] = mi;
25         return ;
26     }
27     int mid = (mi + mx) >> 1;
28     int p1 = 0, p2 = 0;
29     for(int i = lf; i <= rt; ++i) {
30         if(a[i].op == 0) {
31             if(a[i].y <= mid) {
32                 add(a[i].x, 1);
33                 b[++p1] = a[i];
34             } else {
35                 c[++p2] = a[i];
36             }
37         } else {
38             int num = query(a[i].y) - query(a[i].x - 1);
39             if(num >= a[i].z) {
40                 b[++p1] = a[i];
41             } else {
42                 a[i].z -= num;
43                 c[++p2] = a[i];
44             }
```

```
45          }
46      }
47      for(int i = lf; i <= rt; ++i) {
48          if(a[i].op == 0 && a[i].y <= mid) add(a[i].x, -1);
49      }
50      for(int i = 1; i <= p1; ++i) a[lf + i - 1] = b[i];
51      for(int i = 1; i <= p2; ++i) a[lf + p1 + i - 1] = c[i];
52      solve(mi, mid, lf, lf + p1 - 1);
53      solve(mid + 1, mx, lf + p1, rt);
54  }
55
56  int main(){
57      scanf("%d%d", &n, &m);
58      int tot = 0;
59      for(int i = 1; i <= n; ++i) {
60          a[++tot].op = 0;
61          a[tot].x = i;
62          scanf("%d", &a[tot].y);
63      }
64      for(int i = 1; i <= m; ++i) {
65          a[++tot].op = i;
66          scanf("%d%d%d", &a[tot].x, &a[tot].y, &a[tot].z);
67      }
68      solve(-inf, inf, 1, tot);
69      for(int i = 1; i <= m; ++i) printf("%d\n", ans[i]);
70      return 0;
71  }
```

## 8.9 Splay

### 8.9.1 基础用法

第 1 行：1 个正整数 n，表示操作数量，$100 \leq n \leq 200,000$

第 2..n + 1 行：可能包含下面 3 种规则：

1 个字母'I'，紧接着 1 个数字 k，表示插入一个数字 k 到树中，$1 \leq k \leq 1,000,000,000$ ，保证每个 k 都不相同

1 个字母'Q'，紧接着 1 个数字 k。表示询问树中不超过 k 的最大数字

1 个字母'D'，紧接着 2 个数字 a, b ，表示删除树中在区间 $[a, b]$ 的数。

```
1  const int maxn = 2e5 + 5;
2  // 注意inf是否已经足够大
```

```
3  const int inf = 0x3f3f3f3f;
4  struct SplayTree {
5      int son[2];
6      int fa, key, sz, cnt;
7  } S[maxn];
8  int root, tot;
9  void init() {
10     root = tot = 0;
11     S[0].son[0] = S[0].son[1] = 0;
12     S[0].fa = S[0].key = 0;
13     S[0].key = S[0].cnt = 0;
14 }
15 void PushUp(int x) {
16     if(x == 0) return ;
17     SplayTree& now = S[x];
18     now.sz = now.cnt;
19     if(now.son[0]) {
20         SplayTree& ls = S[now.son[0]];
21         now.sz += ls.sz;
22     }
23     if(now.son[1]) {
24         SplayTree& rs = S[now.son[1]];
25         now.sz += rs.sz;
26     }
27 }
28 int GetId(int x) {
29     if(S[S[x].fa].son[1] == x) return 1;
30     return 0;
31 }
32 void SetSon(int x, int fa, int y) {
33     if(x) S[x].fa = fa;
34     if(fa == 0) {
35         root = x;
36         return ;
37     }
38     S[fa].son[y] = x;
39     PushUp(fa);
40 }
41 void Rotate(int x) {
```

```
42      int f = S[x].fa, ff = S[f].fa;
43      int sx = GetId(x), sf = GetId(f);
44      int p = S[x].son[sx ^ 1];
45      SetSon(p, f, sx);
46      SetSon(x, ff, sf);
47      SetSon(f, x, sx ^ 1);
48      PushUp(f); PushUp(x);
49  }
50  void Splay(int x, int y) {
51      while(S[x].fa != y) {
52          int f = S[x].fa, ff = S[f].fa;
53          if(ff != y) {
54              if(GetId(x) == GetId(f)) Rotate(f);
55              else Rotate(x);
56          }
57          Rotate(x);
58      }
59      if(y == 0) root = x;
60  }
61  int Find(int x, int val) {
62      if(x == 0) return 0;
63      SplayTree& now = S[x];
64      if(val < now.key) return Find(now.son[0], val);
65      if(val > now.key) return Find(now.son[1], val);
66      Splay(x, 0);
67      return x;
68  }
69  void Ist(int& x, int& hot, int val, int fa) {
70      if(x == 0) {
71          x = ++tot; hot = x;
72          SplayTree& now = S[x];
73          now.key = val, now.fa = fa;
74          now.son[0] = now.son[1] = 0;
75          now.sz = now.cnt = 1;
76          return ;
77      }
78      SplayTree& now = S[x];
79      if(val == now.key) {
80          ++now.cnt;
```

```
81          hot = x;
82      } else {
83          if(val < now.key) Ist(now.son[0], hot, val, x);
84          else Ist(now.son[1], hot, val, x);
85      }
86      PushUp(x);
87  }
88  void Insert(int val ) {
89      int hot;
90      Ist(root, hot, val, 0);
91      Splay(hot, 0);
92  }
93  void DelRoot() {
94      SplayTree& now = S[root];
95      if(now.cnt > 1) {
96          --now.cnt;
97          --now.sz;
98          return ;
99      }
100     if(now.son[0] == 0) {
101         S[now.son[1]].fa = 0;
102         root = now.son[1];
103         return ;
104     }
105     int cur = S[root].son[0];
106     while(S[cur].son[1]) cur = S[cur].son[1];
107     Splay(cur, root);
108     S[cur].son[1] = S[root].son[1];
109     root = cur; S[cur].fa = 0;
110     if(S[root].son[1]) S[S[root].son[1]].fa = root;
111     PushUp(root);
112 }
113 void Del(int val) {
114     int k = Find(root, val);
115     if(k) DelRoot();
116 }
117 int Kth(int x, int k) {
118     if(x == 0) return 0;
119     SplayTree& now = S[x];
```

```
120     int lt = S[now.son[0]].sz;
121     int le = lt + now.cnt;
122     if(k <= lt) return Kth(now.son[0], k);
123     if(k > le) return Kth(now.son[1], k - le);
124     Splay(x, 0);
125     return x;
126 }
127 int Rank(int x, int val) {
128     if(x == 0) return 0;
129     SplayTree& now = S[x];
130     int lt = S[now.son[0]].sz;
131     int le = lt + now.cnt;
132     if(now.key > val) return Rank(now.son[0], val);
133     if(now.key < val) return le + Rank(now.son[1], val);
134     return lt + 1;
135 }
136 int Prec(int x, int val) {
137     if(x == 0) return 0;
138     SplayTree& now = S[x];
139     if(val <= now.key) return Prec(now.son[0], val);
140     int ret = Prec(now.son[1], val);
141     if(ret == 0) ret = x;
142     return ret;
143 }
144 int Succ(int x, int val) {
145     if(x == 0) return 0;
146     SplayTree& now = S[x];
147     if(val >= now.key) return Succ(now.son[1], val);
148     int ret = Succ(now.son[0], val);
149     if(ret == 0) ret = x;
150     return ret;
151 }
152 void DelSec(int a, int b) {
153     Insert(a); Insert(b);
154     int lst = Prec(root, a);
155     int nxt = Succ(root, b);
156     Splay(lst, 0); Splay(nxt, lst);
157     S[nxt].son[0] = 0;
158 }
```

```
159 char s[55];
160
161 inline int read() {
162     int ret = 0;
163     char ch = getchar();
164     while(ch < '0' || ch > '9') ch = getchar();
165     while(ch >= '0' && ch <= '9'){
166         ret = ret*10 + ch - '0';
167         ch = getchar();
168     }
169     return ret;
170 }
171
172 int main(){
173     init();
174     Insert(-inf);
175     Insert(inf);
176
177     int n = read();
178     for(int cs = 1; cs <= n; ++cs) {
179         scanf("%s", s);
180         if(s[0] == 'I') {
181             int k = read();
182             Insert(k);
183         } else if(s[0] == 'Q') {
184             int k = read();
185             int id = Find(root, k);
186             if(id){
187                 printf("%d\n", k);
188                 continue;
189             }
190             id = Prec(root, k);
191             printf("%d\n", S[id].key);
192         } else {
193             int a = read();
194             int b = read();
195             DelSec(a, b);
196         }
197     }
```

```
198     return 0;
199 }
```

### 8.9.2 区间翻转

您需要维护一个序列，其中需要提供以下操作：

翻转一个区间，例如原有序列是 $5,4,3,2,1$ ，翻转区间是 $[2,4]$ 的话，结果是 $5,2,3,4,1$ 。

第一行为 $n,m$

n 表示初始序列有 n 个数，这个序列依次是 $1,2,\cdots,n-1,n$

m 表示翻转操作次数。

接下来 m 行每行两个数 $[l,r]$ ，表示翻转区间 $[l,r]$，数据保证 $1 \leq l \leq r \leq n$ 。

```
1  const int maxn = 1e5 + 5;
2  const int inf = 0x3f3f3f3f;
3  struct SplayTree {
4      int son[2];
5      int fa, key, sz;
6      int lazy;
7  } S[maxn];
8  int root, tot;
9  void init() {
10     root = tot = 0;
11     S[0].son[0] = S[0].son[1] = 0;
12     S[0].fa = S[0].key = 0;
13     S[0].sz = S[0].lazy = 0;
14 }
15 int AddNode(int key, int fa) {
16     ++tot;
17     S[tot].son[0] = S[tot].son[1] = 0;
18     S[tot].fa = fa; S[tot].key = key;
19     S[tot].lazy = 0; S[tot].sz = 1;
20     return tot;
21 }
22 void PushUp(int x) {
23     if(x == 0) return ;
24     SplayTree& now = S[x];
25     SplayTree& ls = S[now.son[0]];
26     SplayTree& rs = S[now.son[1]];
27     now.sz = 1 + ls.sz + rs.sz;
28 }
```

```
29 int Build(int lf, int rt, int fa) {
30     if(lf > rt) return 0;
31     int mid = (lf + rt) >> 1;
32     int id = AddNode(mid, fa);
33     SplayTree& now = S[id];
34     now.son[0] = Build(lf, mid - 1, id);
35     now.son[1] = Build(mid + 1, rt, id);
36     PushUp(id);
37     return id;
38 }
39 int Judge(int x) {
40     if(S[S[x].fa].son[1] == x) return 1;
41     return 0;
42 }
43 void SetSon(int x, int fa, int id) {
44     if(x) S[x].fa = fa;
45     if(fa == 0) {
46         root = x;
47         return ;
48     }
49     S[fa].son[id] = x;
50     PushUp(fa);
51 }
52 void PushDown(int x) {
53     SplayTree& now = S[x];
54     if(now.lazy == 0) return ;
55     now.lazy = 0;
56     S[now.son[0]].lazy ^= 1;
57     S[now.son[1]].lazy ^= 1;
58     swap(now.son[0], now.son[1]);
59 }
60 void Rotate(int x) {
61     int f = S[x].fa, ff = S[f].fa;
62     int sx = Judge(x), sf = Judge(f);
63     int p = S[x].son[sx ^ 1];
64     SetSon(p, f, sx);
65     SetSon(x, ff, sf);
66     SetSon(f, x, sx ^ 1);
67     PushUp(f); PushUp(x);
```

```
68  }
69  void Splay(int x, int y) {
70      while(S[x].fa != y) {
71          int f = S[x].fa, ff = S[f].fa;
72          if(ff != y) {
73              if(Judge(x) == Judge(f)) Rotate(f);
74              else Rotate(x);
75          }
76          Rotate(x);
77      }
78      if(y == 0) root = x;
79  }
80  int Find(int x, int k) {
81      if(x == 0) return 0;
82      SplayTree& now = S[x];
83      PushDown(x);
84      int lt = S[now.son[0]].sz;
85      int le = lt + 1;
86      if(k <= lt) return Find(now.son[0], k);
87      if(k > le) return Find(now.son[1], k - le);
88      return x;
89  }
90  void Filp(int lf, int rt) {
91      int lst = Find(root, lf - 1 + 1);
92      int nxt = Find(root, rt + 1 + 1);
93      Splay(lst, 0);
94      Splay(nxt, lst);
95      SplayTree& now = S[S[nxt].son[0]];
96      now.lazy ^= 1;
97  }
98  int n, m, cnt = 0;
99  void solve(int x) {
100     if(x == 0) return ;
101     SplayTree& now = S[x];
102     PushDown(x);
103     if(now.key > 0) solve(now.son[0]);
104     if(now.key > 0 && now.key < n + 1) {
105         if(++cnt > 1) printf(" ");
106         printf("%d", S[x].key);
```

```
107     }
108     if(now.key < n + 1) solve(now.son[1]);
109 }
110 int main(){
111     init();
112     scanf("%d%d", &n, &m);
113     root = Build(0, n + 1, 0);
114     for(int cm = 1; cm <= m; ++cm) {
115         int lf, rt; scanf("%d%d", &lf, &rt);
116         Filp(lf, rt);
117     }
118     solve(root); puts("");
119     return 0;
120 }
```

### 8.9.3 区间加减，区间第 K 大

插入、区间加、区间减、区间第 k 大

```
1  typedef long long ll;
2  const int maxn = 1e5 + 5;
3  const int inf = 0x3f3f3f3f;
4  struct SplayTree {
5      int son[2], fa;
6      ll sz, cnt, key, lazy;
7      void pt() {
8          printf("Son: %d %d, Fa %d\n", son[0], son[1], fa);
9          printf("sz: %lld cnt: %lld key: %lld lazy %lld\n", sz, cnt, key,
               lazy);
10     }
11 } S[maxn];
12 int root, tot;
13 void init() {
14     root = tot = 0;
15     S[0].son[0] = S[0].son[1] = S[0].fa = 0;
16     S[0].key = S[0].lazy = 0;
17     S[0].sz = S[0].cnt = 0;
18 }
19 void PushUp(int x) {
20     if(x == 0) return ;
```

```
21      SplayTree& now = S[x];
22      now.sz = now.cnt;
23      if(now.son[0]) {
24          SplayTree& ls = S[now.son[0]];
25          now.sz += ls.sz;
26      }
27      if(now.son[1]) {
28          SplayTree& rs = S[now.son[1]];
29          now.sz += rs.sz;
30      }
31  }
32  void PushDown(int x) {
33      if(x == 0) return ;
34      SplayTree& now = S[x];
35      if(now.lazy == 0) return ;
36      if(now.son[0]) {
37          SplayTree& ls = S[now.son[0]];
38          ls.key += now.lazy;
39          ls.lazy += now.lazy;
40      }
41      if(now.son[1]) {
42          SplayTree& rs = S[now.son[1]];
43          rs.key += now.lazy;
44          rs.lazy += now.lazy;
45      }
46      now.lazy = 0;
47  }
48  int Judge(int x) {
49      if(S[S[x].fa].son[1] == x) return 1;
50      return 0;
51  }
52  void SetSon(int x, int fa, int y) {
53      if(x) S[x].fa = fa;
54      if(fa == 0) {
55          root = x;
56          return ;
57      }
58      S[fa].son[y] = x;
59      PushUp(fa);
```

```
60 }
61 void Rotate(int x) {
62     int f = S[x].fa, ff = S[f].fa;
63     int sx = Judge(x), sf = Judge(f);
64     int p = S[x].son[sx ^ 1];
65     SetSon(p, f, sx);
66     SetSon(x, ff, sf);
67     SetSon(f, x, sx ^ 1);
68     PushUp(f); PushUp(x);
69 }
70 void Splay(int x, int y) {
71     while(S[x].fa != y) {
72         int f = S[x].fa, ff = S[f].fa;
73         if(ff != y) {
74             if(Judge(x) == Judge(f)) Rotate(f);
75             else Rotate(x);
76         }
77         Rotate(x);
78     }
79     if(y == 0) root = x;
80 }
81 void Ist(int& x, int& hot, ll val, int fa) {
82     if(x == 0) {
83         x = ++tot;
84         hot = x;
85         SplayTree& now = S[x];
86         now.key = val; now.fa = fa;
87         now.son[0] = now.son[1] = 0;
88         now.sz = now.cnt = 1; now.lazy = 0;
89         return ;
90     }
91     PushDown(x);
92     SplayTree& now = S[x];
93     if(val == now.key) {
94         ++now.cnt;
95         hot = x;
96     } else {
97         if(val < now.key) Ist(now.son[0], hot, val, x);
98         else Ist(now.son[1], hot, val, x);
```

```
99      }
100     PushUp(x);
101 }
102 void Insert(ll val) {
103     int hot;
104     Ist(root, hot, val, 0);
105     Splay(hot, 0);
106 }
107 int Kth(int x, int k) {
108     if(x == 0) return 0;
109     PushDown(x);
110     SplayTree& now = S[x];
111     ll lt = S[now.son[0]].sz;
112     ll le = lt + now.cnt;
113     if(k <= lt) return Kth(now.son[0], k);
114     if(k > le) return Kth(now.son[1], k - le);
115     Splay(x, 0);
116     return x;
117 }
118 int Succ(int x, ll val) {
119     if(x == 0) return 0;
120     PushDown(x);
121     SplayTree& now = S[x];
122     if(val >= now.key) return Succ(now.son[1], val);
123     int ret = Succ(now.son[0], val);
124     if(ret == 0) ret = x;
125     return ret;
126 }
127 void Update(ll val) {
128     if(root == 0) return ;
129     SplayTree& now = S[root];
130     now.key += val;
131     now.lazy += val;
132 }
133 ll mi, ans;
134 void clr() {
135     int id = Succ(root, mi - 1LL);
136     if(id == 0) {
137         ans += S[root].sz;
```

```
138         root = 0;
139         return ;
140     }
141     Splay(id, 0);
142     ans += S[S[id].son[0]].sz;
143     S[id].son[0] = 0;
144     PushUp(id);
145 }
146 char opt[5];
147 int main(){
148     int n; scanf("%d%lld", &n, &mi);
149     init(); ans = 0;
150     for(int cn = 1; cn <= n; ++cn) {
151         scanf("%s", opt);
152         if(opt[0] == 'I') {
153             ll k; scanf("%lld", &k);
154             if(k >= mi) Insert(k);
155         } else if(opt[0] == 'A') {
156             ll k; scanf("%lld", &k);
157             Update(k);
158         } else if(opt[0] == 'S') {
159             ll k; scanf("%lld", &k);
160             Update(-k);
161         } else {
162             int k; scanf("%d", &k);
163             int num = S[root].sz;
164             if(num < k) puts("-1");
165             else printf("%lld\n", S[Kth(root, num - k + 1)].key);
166         }
167         clr();
168     }
169     printf("%lld\n", ans);
170     return 0;
171 }
```

## 8.10 Link-Cut Tree

```
1 const int maxn = 3e5 + 5;
2 struct LCT {
```

```
3     int tp, st[maxn];
4     int fa[maxn], ch[maxn][2];
5     bool lazy[maxn];
6     int mx[maxn], val[maxn];
7     void init(int x, int y) {
8         fa[x] = ch[x][0] = ch[x][1] = 0;
9         lazy[x] = false;
10        mx[x] = x; val[x] = y;
11    }
12    bool is_root(int x) {
13        return ch[fa[x]][0] != x && ch[fa[x]][1] != x;
14    }
15    void push_up(int x) {
16        int lc = ch[x][0], rc = ch[x][1];
17        mx[x] = x;
18        if (val[mx[x]] < val[mx[lc]]) mx[x] = mx[lc];
19        if (val[mx[x]] < val[mx[rc]]) mx[x] = mx[rc];
20    }
21    void push_down(int x) {
22        if (!lazy[x]) return ;
23        int lc = ch[x][0], rc = ch[x][1];
24        lazy[x] = !lazy[x];
25        lazy[lc] = !lazy[lc];
26        lazy[rc] = !lazy[rc];
27        swap(ch[x][0], ch[x][1]);
28    }
29    void rotate(int x) {
30        int y = fa[x], z = fa[y], lc, rc;
31        if (ch[y][0] == x) lc = 0;
32        else lc = 1;
33        rc = lc ^ 1;
34        if (!is_root(y)) {
35            if (ch[z][0] == y) ch[z][0] = x;
36            else ch[z][1] = x;
37        }
38        fa[x] = z; fa[y] = x; fa[ch[x][rc]] = y;
39        ch[y][lc] = ch[x][rc]; ch[x][rc] = y;
40        push_up(y); push_up(x);
41    }
```

```
42      void splay(int x) {
43          tp = 0; st[tp++] = x;
44          for (int i = x; !is_root(i); i = fa[i]) st[tp++] = fa[i];
45          while (tp > 0) push_down(st[--tp]);
46          while (!is_root(x)) {
47              int y = fa[x], z = fa[y];
48              if (!is_root(y)) {
49                  if ((ch[y][0] == x) == (ch[z][0] == y)) rotate(x);
50                  else rotate(y);
51              }
52              rotate(x);
53          }
54      }
55      void access(int x) {
56          for (int t = 0; x != 0; t = x, x = fa[x]) {
57              splay(x); ch[x][1] = t; push_up(x);
58          }
59      }
60      void make_root(int x) {
61          access(x); splay(x); lazy[x] = !lazy[x];
62      }
63      void link(int x, int y) {
64          make_root(x); fa[x] = y;
65      }
66      void cut(int x, int y) {
67          make_root(x); access(y); splay(y);
68          ch[y][0] = fa[x] = 0; push_up(y);
69      }
70      int query(int x, int y) {
71          make_root(x); access(y); splay(y);
72          int ret = mx[y];
73          while (ch[y][0] != 0) {
74              y = ch[y][0];
75              if (val[ret] < val[mx[y]]) val[ret] = mx[y];
76          }
77          if (x != y) return -1;
78          return ret;
79      }
80  } splay;
```

## 8.11 Euler Tour Tree

```
struct Euler_Tour_Tree{
    stack<int> sta;
    int tot;
    int tagp[300005]; // 记录ETT的点的出现第一次
    map<int, int> e[300005]; // 记录ETT的边出现
    struct node {
        int ls, rs, fa, data, size;
        int rd;
        bool tag, stag; // 是否有第一次出现的点
    } v[600005];
    int new_node(int pt){
        int x;
        if (!sta.empty()) x = sta.top(), sta.pop();
        else x = ++tot;
        v[x].fa = v[x].ls = v[x].rs = 0;
        v[x].tag = v[x].stag = 0;
        v[x].size = 1;
        v[x].data = pt;
        v[x].rd = rand();
        return x;
    }
    void del_node(int x){
        sta.push(x);
        return;
    }
    void push_up(int rt){
        v[rt].stag = v[v[rt].ls].stag|v[v[rt].rs].stag|v[rt].tag;
        v[rt].size = v[v[rt].ls].size + v[v[rt].rs].size + 1;
    }
    int mer_ge(int x, int y){
        if (!x || !y) return x | y;
        if (v[x].rd <= v[y].rd){
            v[x].rs = mer_ge(v[x].rs,y);
            v[v[x].rs].fa = x;
            push_up(x);
            return x;
        } else if (v[x].rd >= v[y].rd) {
            v[y].ls = mer_ge(x, v[y].ls);
```

```
39              v[v[y].ls].fa = y;
40              push_up(y);
41              return y;
42          }
43      }
44      int get_root(int x){
45          while (v[x].fa) x = v[x].fa;
46          return x;
47      }
48      void split(int x, int d, int &r1, int &r2) {
49          int s1 = v[x].ls, s2 = v[x].rs;
50          if (d == -1){
51              v[x].rs = 0;
52              push_up(x);
53              s1 = x;
54              v[s2].fa = 0;
55          }
56          if (d == 1) {
57              v[x].ls = 0;
58              push_up(x);
59              s2 = x;
60              v[s1].fa = 0;
61          }
62          if (d != 1 && d != -1) v[s1].fa = v[s2].fa = 0;
63          while (v[x].fa) {
64              int p = v[x].fa;
65              if (v[p].ls == x) {
66                  v[p].ls = s2;
67                  v[s2].fa = p;
68                  v[s1].fa = 0;
69                  s2 = p;
70              }
71              if (v[p].rs == x) {
72                  v[p].rs = s1;
73                  v[s1].fa = p;
74                  v[s2].fa = 0;
75                  s1 = p;
76              }
77              push_up(x = p);
```

```
78          }
79          r1 = s1; r2 = s2;
80      }
81      int get_p(int x){
82          if (e[x].empty()) return 0;
83          return e[e[x].begin()->first][x];
84      }
85      void link(int x, int y){
86          int p1 = new_node(y), p2 = new_node(x);
87          if (tagp[x]==-1){
88              v[p2].tag = v[p2].stag = 1;
89              tagp[x] = p2;
90          }
91          if (tagp[y] == -1){
92              v[p1].tag = v[p1].stag = 1;
93              tagp[y] = p1;
94          }
95          int t1, t2, t3;
96          if (e[x].empty()) t1 = t3 = 0;
97          else split(e[x].begin()->second, 1, t1, t3);
98          if (e[y].empty()) t2 = 0;
99          else {
100             int tt1, tt2;
101             split(e[y].begin()->second, 1, tt1, tt2);
102             t2 = mer_ge(tt2, tt1);
103         }
104         e[x][y] = p1;
105         e[y][x] = p2;
106         mer_ge(mer_ge(mer_ge(t1, p1), t2), mer_ge(p2, t3));
107     }
108     void cut(int x,int y) {
109         int p1 = e[x][y], p2 = e[y][x];
110         e[x].erase(e[x].find(y));
111         e[y].erase(e[y].find(x));
112         int t11, t12;
113         split(p1, 0, t11, t12);
114         if(get_root(p2) == t11){
115             int t21, t22;
116             split(p2, 0, t21, t22);
```

```
117             mer_ge(t21, t12);
118         } else {
119             int t21, t22;
120             split(p2, 0, t21, t22);
121             mer_ge(t11, t22);
122         }
123         if(v[p1].tag){
124             if(e[y].empty()) tagp[y] = -1;
125             else {
126                 tagp[y] = get_p(y);
127                 v[tagp[y]].tag = 1;
128                 for(int i = tagp[y]; i; i = v[i].fa) push_up(i);
129             }
130         }
131         if (v[p2].tag){
132             if (e[x].empty()) tagp[x] = -1;
133             else {
134                 tagp[x] = get_p(x);
135                 v[tagp[x]].tag = 1;
136                 for (int i = tagp[x]; i; i = v[i].fa) push_up(i);
137             }
138         }
139         del_node(p1);
140         del_node(p2);
141     }
142     bool islink(int x, int y) {
143         int p1 = get_p(x), p2 = get_p(y);
144         if (!p1 || !p2) return false;
145         return get_root(p1) == get_root(p2);
146     }
147 } ETT;
```

## 8.12 李超树（二维平面维护直线）

```
1 const int maxn = 1e5 + 5;
2 int tree[maxn<<2];
3 char s[15];
4 double k[maxn], b[maxn];
5 int tot;
```

```
6  double cal(int id, double x) {
7      double ret = k[id]*x + b[id];
8      return ret;
9  }
10 int sgn(double x) {
11     if(fabs(x) < eps) return 0;
12     if(x < 0) return -1;
13     return 1;
14 }
15 double getPos(double k1, double b1, double k2, double b2) {
16     double ret = (b2 - b1)/(k1 - k2);
17     return ret;
18 }
19 void ist(int x, int lf, int rt, int id) {
20     if(tree[x] == 0) {
21         tree[x] = id;
22         return ;
23     }
24     double vl1 = cal(id, (double)lf), vl2 = cal(tree[x], (double)lf);
25     double vr1 = cal(id, (double)rt), vr2 = cal(tree[x], (double)rt);
26     int dt1 = sgn(vl1 - vl2), dt2 = sgn(vr1 - vr2);
27     if(dt1 <= 0 && dt2 <= 0) return ;
28     if(dt1 >= 0 && dt2 >= 0) {
29         tree[x] = id;
30         return ;
31     }
32     int mid = (lf + rt) >> 1;
33     double dm = mid;
34     double pos = getPos(k[id], b[id], k[tree[x]], b[tree[x]]);
35     if(dt1 >= 0) {
36         if(sgn(pos - dm) <= 0) {
37             ist(x << 1, lf, mid, id);
38             tree[x] = tree[x];
39         } else {
40             ist(x << 1 | 1, mid + 1, rt, tree[x]);
41             tree[x] = id;
42         }
43     } else {
44         if(sgn(pos - dm) <= 0) {
```

```
45              ist(x << 1, lf, mid, tree[x]);
46              tree[x] = id;
47          } else {
48              ist(x << 1 | 1, mid + 1, rt, id);
49              tree[x] = tree[x];
50          }
51      }
52  }
53  double query(int x, int lf, int rt, double pos) {
54      double ret = 0.0;
55      if(tree[x] != 0) ret = max(ret, cal(tree[x], pos));
56      if(lf == rt) return ret;
57      int mid = (lf + rt) >> 1;
58      double dm = mid;
59      if(sgn(dm - pos) >= 0) ret = max(ret, query(x << 1, lf, mid, pos));
60      else ret = max(ret, query(x << 1 | 1, mid + 1, rt, pos));
61      return ret;
62  }
```

## 8.13 左偏树（可并堆）(log(n) 时间内合并两个堆)

```
1   template<class T>
2   struct LeftistTree {
3       static const int NodeNum = 1e6 + 5;
4       int tot;
5       struct Data {
6           T key;
7           int l, r, dist;
8       } tr[NodeNum << 1];
9       void Init() {
10          tot = 0;
11      }
12      LeftistTree() {
13          Init();
14      }
15      int New(T key) {
16          ++tot;
17          tr[tot].l = tr[tot].r = 0;
18          tr[tot].key = key;
```

```
19          tr[tot].dist = 0;
20          return tot;
21      }
22      int Merge(int x, int y) {
23          if(x == 0) return y;
24          if(y == 0) return x;
25          if(tr[y].key > tr[x].key) swap(x, y);
26          tr[x].r = Merge(tr[x].r, y);
27          if(tr[tr[x].l].dist < tr[tr[x].r].dist)
28              swap(tr[x].l, tr[x].r);
29          tr[x].dist = tr[tr[x].r].dist + 1;
30          return x;
31      }
32      void Push(int &x, T key) {
33          x = Merge(x, New(key));
34      }
35      T Pop(int& x) {
36          T ret = tr[x].key;
37          x = Merge(tr[x].l, tr[x].r);
38          return ret;
39      }
40      T Top(int& x) {
41          return tr[x].key;
42      }
43  };
44  LeftistTree<int> Tree;
```

## 8.14 哈希表

```
1   template <typename T1, typename T2>
2   class Hash {
3   public:
4       static const int HashMod = 7679977;
5       // 99991, 3000017, 7679977, 19260817
6       int tp, st[HashMod + 5];
7       bool vis[HashMod + 5];
8       T1 h[HashMod + 5];
9       T2 val[HashMod + 5];
10      int locate(const T1& x) const {
```

```
11          int p = x%HashMod;
12          while(vis[p] && h[p] != x) {
13              if(++p == HashMod) p = 0;
14          }
15          return p;
16      }
17      Hash() {
18          tp = 0;
19          memset(vis, false, sizeof(false));
20      }
21      void ist(const T1& x, const T2& y) {
22          const int p = locate(x);
23          assert(!vis[p] || h[p] == x);
24          if (!vis[p]) {
25              h[p] = x; vis[p] = true;
26              val[p] = y;
27              st[++tp] = p;
28          } else if (h[p] == x) {
29              val[p] = y;
30          } else {
31              assert(false);
32          }
33      }
34      bool get(const T1& x, T2& y) const {
35          const int p = locate(x);
36          if(vis[p] && h[p] == x) {
37              y = val[p];
38              return true;
39          }
40          return false;
41      }
42      void prt() {
43          for (int i = 1; i <= tp; ++i) {
44              if (i > 1) cout << " ";
45              cout << h[st[i]];
46          }
47          cout << endl;
48      }
49      void clr() {
```

```
50          while (tp > 0) vis[st[tp--]] = false;
51      }
52  };
53  Hash<int, int> H;
```

# 9 动态规划

## 9.1 最长不降（上升）子序列 O(nlogn)

### 9.1.1 模板

```
1  const int maxn = 1e5 + 5;
2  int a[maxn], b[maxn];
3  int LIS(int n){
4      int tot = 0, ret = 0;
5      for(int i = 0; i < n; ++i){
6          int pos = lower_bound(b, b + tot, a[i]) - b;
7          if(pos == tot) b[tot++] = a[i];
8          else b[pos] = a[i];
9          ret = max(ret, pos + 1);
10     }
11     return ret;
12 }
```

### 9.1.2 最长上升子序列计数（离散化 + 树状数组）

```
1  const int mod = 1e9 + 7;
2  const int maxn = 5e4 + 5;
3  int a[maxn];
4  vector<int> b;
5  unordered_map<int, int> id;
6  int len, num;
7  int mx[maxn << 1], cnt[maxn << 1];
8  // 注意树状数组开 2 倍
9  #define lowbit(x) (x&(-x))
10 void query(int x) {
11     while(x > 0) {
12         if(mx[x] > len) {
13             len = mx[x];
```

```
14              num = cnt[x];
15          } else if(mx[x] == len) {
16              num = (cnt[x] + num)%mod;
17          }
18          x -= lowbit(x);
19      }
20  }
21  void update(int x) {
22      while(x < maxn) {
23          if(mx[x] < len) {
24              mx[x] = len;
25              cnt[x] = num;
26          } else if(mx[x] == len) {
27              cnt[x] = (cnt[x] + num)%mod;
28          }
29          x += lowbit(x);
30      }
31  }
32  int main() {
33      int n; scanf("%d", &n);
34      for(int i = 1; i <= n; ++i) {
35          scanf("%d", &a[i]);
36          b.push_back(a[i]);
37          b.push_back(a[i] - 1);
38      }
39      sort(b.begin(), b.end());
40      b.erase(unique(b.begin(), b.end()), b.end());
41      int sz = b.size();
42      for(int i = 0; i < sz; ++i) id[b[i]] = i + 1;
43      for(int i = 1; i <= n; ++i) a[i] = id[a[i]];
44      for(int i = 1; i <= n; ++i) {
45          len = 0, num = 1;
46          query(a[i] - 1);
47          ++len;
48          update(a[i]);
49      }
50      len = 0, num = 1;
51      query(maxn - 1);
52      printf("%d\n", num);
```

```
53      return 0;
54  }
```

## 9.2 编辑距离

dp[i][j] 记录 s1 的前 i 个字符转换到 s2 的前 j 个字符的最小编辑距离

```
1   int dp[maxn][maxn];
2   int CompareString(char *s1, char *s2) {
3       memset(dp, 0, sizeof(dp));
4       int m = strlen(s1), n = strlen(s2);
5       for(int i = 0; i <= m; ++i) dp[i][0] = i;
6       for(int j = 0; j <= n; ++j) dp[0][j] = j;
7       for(int i = 1; i <= m; ++i){
8           for(int j = 1; j <= n; ++j){
9               int cost = 0;
10              if(s1[i - 1] == s2[j - 1]) cost = 0;
11              else cost = 1;
12              dp[i][j] = min(min(dp[i - 1][j - 1] + cost, dp[i - 1][j] + 1),
                    dp[i][j - 1] + 1);
13          }
14      }
15      return dp[m][n];
16  }
```

## 9.3 计算 hx 进制下所有位 num 出现的次数

```
1   ll tot[20];
2   // n 数字最大位数, hx 进制。 比如 若最大数字为 1e18, 则 n = 19
3   void init(int n, ll hx) {
4       tot[0] = 0;
5       ll bs = 1LL;
6       for(int i = 1; i < n; ++i) {
7           tot[i] = tot[i - 1]*hx + bs;
8           bs *= hx;
9       }
10      for(int i = 0; i < n; ++i) {
11          printf("%d: %lld\n", i, tot[i]);
12      }
13  }
```

```
14 // 计算 hx 进制下 [0, x] ，所有位（考虑算上前导零共 n 位） num 出现的次数
       （lz 为是否考虑前导零）
15 ll cnt(int n, ll x, ll num, ll hx, bool lz) {
16     ll ret = 0, dig = 0, bs = 1, tail = 0, xx = x;
17     int len = 0;
18     if(lz) {
19         // 计算前导零
20         while(len < n) {
21             dig = x%hx;
22             x /= hx;
23             if(dig > num) ret += bs + dig*tot[len];
24             else if(dig == num) ret += tail + 1LL + dig*tot[len];
25             else ret += dig*tot[len];
26             tail += dig*bs;
27             bs *= hx;
28             ++len;
29         }
30     } else {
31         // 不计算前导零
32         while(x > 0) {
33             dig = x%hx;
34             x /= hx;
35             if(dig > num) ret += bs + dig*tot[len];
36             else if(dig == num) ret += tail + 1LL + dig*tot[len];
37             else ret += dig*tot[len];
38             tail += dig*bs;
39             bs *= hx;
40             ++len;
41         }
42         if(num == 0) {
43             bs = 1;
44             while(xx > 0) {
45                 ret -= bs;
46                 xx /= hx;
47                 bs *= hx;
48             }
49         }
50     }
51     return ret;
```

```
52 }
```

## 9.4 期望、概率 dp

[讲解博客链接](http://www.cnblogs.com/CzYoL/p/7220088.html)

【期望 dp】

求解达到某一目标的期望花费：

因为最终的花费无从知晓（不可能从 ∞ 推起），所以期望 dp 需要倒序求解。

设 f[i][j] 表示在 (i, j) 这个状态实现目标的期望值（相当于是差距是多少）。

首先 f[n][m] = 0，在目标状态期望值为 0。

然后 f = ( f × p) + w，f 为上一状态（距离目标更近的那个，倒序），p 为从 f 转移到 f 的概率（则从 f 转移回 f 的概率也为 p），w 为转移的花费。

最后输出初始位置的 f 即可。

特别的，当转移关系不成环时，期望 dp 可以线性递推。

但当转移关系成环时，期望 dp 的最终状态相当于一个已知量，而转移关系相当于一个个方程，可以使用【高斯消元】解决。

【概率 dp】

概率 dp 通常已知初始的状态，然后求解最终达到目标的概率，所以概率 dp 需要顺序求解。

概率 dp 相对简单，当前状态只需加上所有上一状态乘上转移概率即可：f = f i × pi

## 9.5 数位 dp

### 9.5.1 模板

```
1 int a[20];
2 ll dp[20][state]; // 不同题目状态不同
3 ll dfs(int pos, /* state变量 */, bool lead /* 前导零 */, bool limit /* 数
    位上界变量 */) // 不是每个题都要判断前导零
4 {
5     // 递归边界，既然是按位枚举，最低位是 0，那么pos == -1 说明这个数我枚
        举完了
6     if (pos == -1) return 1; /*这里一般返回1，表示你枚举的这个数是合法的，
        那么这里就需要你在枚举时必须每一位都要满足题目条件，也就是说当前枚
        举到pos位，一定要保证前面已经枚举的数位是合法的。不过具体题目不同或
        者写法不同的话不一定要返回1 */
7     // 第二个就是记忆化(在此前可能不同题目还能有一些剪枝)
8     if (!limit && !lead && dp[pos][state] != -1) return dp[pos][state];
9     /*常规写法都是在没有限制的条件记忆化，这里与下面记录状态是对应，具体为
        什么是有条件的记忆化后面会讲*/
```

```
10      int up = limit ? a[pos] : 9; // 根据limit判断枚举的上界up;这个的例子前面用213讲过了
11      ll ans = 0;
12      // 开始计数
13      for (int i = 0; i <= up; i++) // 枚举，然后把不同情况的个数加到ans就可以了
14      {
15          if () ...
16          else if ()...
17          ans += dfs(pos - 1, /* 状态转移 */, lead && i == 0, limit && i == a[pos]) // 最后两个变量传参都是这样写的
18          /* 这里还算比较灵活，不过做几个题就觉得这里也是套路了
19          大概就是说，我当前数位枚举的数是 i，然后根据题目的约束条件分类讨论
20          去计算不同情况下的个数，还有要根据 state 变量来保证 i 的合法性，比如题目
21          要求数位上不能有 62 连续出现,那么就是 state 就是要保存前一位 pre，然后分类，
22          前一位如果是 6 那么这意味就不能是 2，这里一定要保存枚举的这个数是合法 */
23      }
24      // 计算完，记录状态
25      if(!limit && !lead) dp[pos][state] = ans;
26      /* 这里对应上面的记忆化，在一定条件下时记录，保证一致性，当然如果约束条件不需要考虑 lead，这里就是 lead 就完全不用考虑了 */
27      return ans;
28  }
29  ll solve(ll x)
30  {
31      int pos = 0;
32      while (x) // 把数位都分解出来
33      {
34          a[pos++] = x%10; // 个人老是喜欢编号为[0, pos)，看不惯的就按自己习惯来，反正注意数位边界就行
35          x /= 10;
36      }
37      return dfs(pos - 1 /* 从最高位开始枚举 */, /* 一系列状态 */, true, true); // 刚开始最高位都是有限制并且有前导零的，显然比最高位还要高的一位视为 0 嘛
38  }
```

### 9.5.2 数字计数 count

给定两个正整数 a 和 b ，求在 [a, b] 中的所有整数中，每个数码 (digit) 各出现了多少次。

```
#include <iostream>
#include <cstdio>
#include <cmath>
using namespace std;
typedef long long ll;
const int mod = 1e9 + 7;
const int maxn = 1e5 + 5;
ll dp[12][15][15];
int a[15];
ll dfs(int pos, int dig, bool lead, bool limit, int cnt) {
    if(pos == -1) return cnt;
    if(!limit && !lead && dp[dig][pos][cnt] != -1) return dp[dig][pos][cnt];
    int num = a[pos];
    int up = 9;
    if(limit) up = num;
    ll ret = 0;
    for(int i = 0; i <= up; ++i) {
        int nc = cnt;
        if(i == dig) {
            if(dig != 0) ++nc;
            else if(!lead) ++nc;
        }
        ret += dfs(pos - 1, dig, lead && (i == 0), limit && (i == up), nc);
    }
    if(!lead && !limit) dp[dig][pos][cnt] = ret;
    return ret;
}
ll solve(ll x, int dig) {
    int pos = 0;
    while(x > 0) {
        a[pos++] = x%10LL;
        x /= 10LL;
    }
    ll ret = dfs(pos - 1, dig, true, true, 0);
    return ret;
```

```
}
int main() {
    ll a, b;
    for(int i = 0; i < 10; ++i) {
        for(int j = 0; j <= 13; ++j) {
            for(int k = 0; k <= 13; ++k) {
                dp[i][j][k] = -1;
            }
        }
    }
    while(scanf("%lld%lld", &a, &b) != EOF) {
        for(int i = 0; i <= 9; ++i) {
            ll ans = solve(b, i) - solve(a - 1, i);
            if(i > 0) printf(" ");
            printf("%lld", ans);
        }
        puts("");
    }
    return 0;
}
```

## 9.6 括号序列相关

括号序列配对时满足贪心性质，从左向右考虑，能配对的括号直接配对，结果不会变差。

### 9.6.1 最长合法括号序列

```
char str[maxn];
int dp[maxn];
stack<int> s;
int mx, sum;//最长长度，以及最长长度的个数
void LongestRegularBracketSequence(){
    mx = 0, sum = 1;
    int len = strlen(str);
    for(int i = 0; i < len; i++) {
        if(str[i] == '('){
            s.push(i);
            continue;
        }
        if(!s.empty()) {
```

```
14              int now = s.top();
15              s.pop();
16              dp[i] = i - now + 1;
17              if(now) dp[i] += dp[now - 1];
18              if(mx < dp[i]) {
19                  mx = dp[i];
20                  sum = 0;
21              }
22              if(mx == dp[i]) ++sum;
23          }
24      }
25  }
```

### 9.6.2 区间询问最长合法括号序列

```
1   struct SegTree {
2       int lf, rt;
3       int a, b, c;
4   } st[maxn << 2];
5   char s[maxn];
6   void PushUp(int x) {
7       SegTree &now = st[x];
8       SegTree &lc = st[x << 1];
9       SegTree &rc = st[x << 1 | 1];
10      int ok = min(lc.a, rc.b);
11      now.a = lc.a + rc.a - ok;
12      now.b = lc.b + rc.b - ok;
13      now.c = lc.c + rc.c + ok;
14  }
15  void Build(int x, int lf, int rt) {
16      SegTree &now = st[x];
17      now.lf = lf; now.rt = rt;
18      now.a = now.b = now.c = 0;
19      if(lf == rt) {
20          if(s[lf] == '(') ++now.a;
21          else ++now.b;
22          return ;
23      }
24      int mid = (lf + rt) >> 1;
```

```
25      Build(x << 1, lf, mid);
26      Build(x << 1 | 1, mid + 1, rt);
27      PushUp(x);
28  }
29  int qa, qb, qc;
30  void Query(int x, int lf, int rt) {
31      SegTree &now = st[x];
32      if(now.lf >= lf && now.rt <= rt) {
33          int ok = min(qa, now.b);
34          qa = qa + now.a - ok;
35          qb = qb + now.b - ok;
36          qc = qc + now.c + ok;
37          return ;
38      }
39      int mid = (now.lf + now.rt) >> 1;
40      if(lf <= mid) Query(x << 1, lf, rt);
41      if(rt > mid) Query(x << 1 | 1, lf, rt);
42  }
43  int main() {
44      scanf("%s", s + 1);
45      int n = strlen(s + 1);
46      Build(1, 1, n);
47      int m; scanf("%d", &m);
48      for(int im = 1; im <= m; ++im) {
49          int l, r; scanf("%d%d", &l, &r);
50          qa = qb = qc = 0;
51          Query(1, l, r);
52          printf("%d\n", qc*2);
53      }
54      return 0;
55  }
```

## 9.7 斜率优化 dp

### 9.7.1 模板

```
1  // 此处维护下凸壳
2  // 若需要维护上凸壳，则需要改变 judge1(), judge2() 中的不等号方向
3  ll dp[maxn], up[maxn], down[maxn], k[maxn];
4  int mq[maxn];
```

```
5  bool judge1(int x, int y, int z) {
6      ll v1 = up[y] - up[x];
7      ll v2 = (down[y] - down[x])*k[z];
8      if(v1 <= v2) return true;
9      return false;
10 }
11 bool judge2(int x, int y, int z) {
12     ll v1 = (up[y] - up[x])*(down[z] - down[y]);
13     ll v2 = (up[z] - up[y])*(down[y] - down[x]);
14     if(v1 >= v2) return true;
15     return false;
16 }
17 void getZero() {
18
19 }
20 ll getK(int x) {
21     ll ret = 0;
22     return ret;
23 }
24 ll getDp(int x, int y) {
25     ll ret = 0;
26     return ret;
27 }
28 ll getUp(int x) {
29     ll ret = 0;
30     return ret;
31 }
32 ll getDown(int x) {
33     ll ret = 0;
34     return ret;
35 }
36 ll solve(int n) {
37     getZero();
38     int lf = 0, rt = 0;
39     for(int i = 0; i <= n; ++i) {
40         k[i] = getK(i);
41         while(lf + 1 < rt && judge1(mq[lf], mq[lf + 1], i)) ++lf;
42         dp[i] = getDp(mq[lf], i);
43         up[i] = getUp(i); down[i] = getDown(i);
```

```
44          while(lf + 1 < rt && judge2(mq[rt - 2], mq[rt - 1], i)) --rt;
45          mq[rt++] = i;
46      }
47      return dp[n];
48  }
```

### 9.7.2 例题

dp 方程: $dp[i] = mindp[j] + pre[j] - x[i] \times sum[j]) + c[i] + sum[i] \times x[i] - pre[i]$

```
1   const int maxn = 1e6 + 5;
2   ll x[maxn], p[maxn], c[maxn];
3   ll sum[maxn], dp[maxn], pre[maxn];
4   ll up[maxn], down[maxn];
5   int mq[maxn], lf = 0, rt = 1;
6   int judge1(int p1, int p2, ll val) {
7       ll dt = up[p2] - up[p1] - (down[p2] - down[p1])*val;
8       if(dt == 0) return 0;
9       if(dt < 0) return -1;
10      return 1;
11  }
12  int judge2(int p1, int p2, int p3) {
13      ll dt = (up[p3] - up[p2])*(down[p2] - down[p1]) - (up[p2] - up[p1])*(
            down[p3] - down[p2]);
14      if(dt == 0) return 0;
15      if(dt < 0) return -1;
16      return 1;
17  }
18  int main(){
19      int n; scanf("%d", &n);
20      for(int i = 1; i <= n; ++i) scanf("%lld%lld%lld", &x[i], &p[i], &c[i])
            ;
21      for(int i = 1; i <= n; ++i) {
22          sum[i] = sum[i - 1] + p[i];
23          pre[i] = pre[i - 1] + p[i]*x[i];
24      }
25      for(int i = 1; i <= n; ++i) {
26          while(rt - lf >= 2 && judge1(mq[lf], mq[lf + 1], x[i]) < 0) ++lf;
27          int id = mq[lf];
28          dp[i] = dp[id] + (sum[i] - sum[id])*x[i] - (pre[i] - pre[id]) + c[
```

```
            i];
29          up[i] = dp[i] + pre[i]; down[i] = sum[i];
30          while(rt - lf >= 2 && judge2(mq[rt - 2], mq[rt - 1], i) < 0) --rt;
31          mq[rt++] = i;
32      }
33      printf("%lld\n", dp[n]);
34      return 0;
35  }
```

## 9.8 四边形不等式

### 9.8.1 基本理论

如果对于任意的 $a_1 \le a_2 < b_1 \le b_2$ 有 $m[a_1,b_1]+m[a_2,b_2] \le m[a_1,b_2]+m[a_2,b_1]$，那么 $m[i,j]$ 满足四边形不等式。

设 $m[i,j]$ 表示动态规划的状态量。

$m[i,j]$ 有类似如下的状态转移方程：

$m[i,j] = min\{m[i,k]+m[k,j]\}(i \le k \le j)$

m 满足四边形不等式是适用这种优化方法的必要条件

对于一道具体的题目，我们首先要证明它满足这个条件，一般来说用数学归纳法证明，根据题目的不同而不同。

通常的动态规划的复杂度是 $O(n^3)$，我们可以优化到 $O(n^2)$

定义 s(i,j) 为函数 m(i,j) 对应的使得 m(i,j) 取得最小值的 k 值。

我们可以证明，s[i,j-1] s[i,j] s[i+1,j]

那么改变状态转移方程为：

m[i,j]=minm[i,k]+m[k,j](s[i,j-1] k s[i+1,j])

复杂度分析：不难看出，复杂度决定于 s 的值，以求 m[i,i+L] 为例，

(s[2,L+1]-s[1,L])+(s[3,L+2]-s[2,L+1])…+(s[n-L+1,n]-s[n-L,n-1])=s[n-L+1,n]-s[1,L] n 所以总复杂度是 O(n)

### 9.8.2 优化环形石子合并 ($O(n^3)$ To $O(n^2)$)

N 堆石子摆成一个环。现要将石子有次序地合并成一堆。

规定每次只能选相邻的 2 堆石子合并成新的一堆，并将新的一堆石子数记为该次合并的代价。

计算将 N 堆石子合并成一堆的最小代价。

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  const int inf = 0x3f3f3f3f;
5  const ll linf = 0x3f3f3f3f3f3f3f3f;
```

```
6  const int mod = 1e9 + 7;
7  const int maxn = 2e3 + 5;
8  ll a[maxn], pre[maxn];
9  ll m[maxn][maxn], w[maxn][maxn];
10 int s[maxn][maxn], n;
11 void init() {
12     for(int i = 1; i <= n; ++i) a[i + n] = a[i];
13     for(int i = 1; i <= 2*n; ++i) {
14         pre[i] = pre[i - 1] + a[i];
15         for(int j = 1; j <= 2*n; ++j) {
16             if(j > i) w[j][i] = linf;
17             else w[j][i] = pre[i] - pre[j - 1];
18             m[i][j] = linf;
19         }
20     }
21     for(int i = 1; i <= 2*n; ++i) {
22         m[i][i] = 0;
23         s[i][i] = i;
24     }
25 }
26 ll solve() {
27     for(int len = 2; len <= n; ++len) {
28         for(int lf = 1; lf <= 2*n; ++lf) {
29             int rt = lf + len - 1;
30             if(rt >= 2*n) break;
31             for(int mid = s[lf][rt - 1]; mid <= s[lf + 1][rt]; ++mid) {
32                 ll now = m[lf][mid] + m[mid + 1][rt] + w[lf][rt];
33                 if(now > m[lf][rt]) continue;
34                 s[lf][rt] = mid; m[lf][rt] = now;
35             }
36         }
37     }
38     ll ret = linf;
39     for(int i = 1; i <= n; ++i) ret = min(ret, m[i][i + n - 1]);
40     return ret;
41 }
42 int main() {
43     scanf("%d", &n);
44     for(int i = 1; i <= n; ++i) scanf("%lld", &a[i]);
```

```
45      init();
46      ll ans = solve();
47      printf("%lld\n", ans);
48      return 0;
49  }
```

### 9.9 最大 m 子段和

```
1   const int maxn = 5e3 + 5;
2   ll a[maxn], dp[2][maxn];
3   ll MaxSum(int m, int n){
4       memset(dp, 0, sizeof(dp));
5       int now = 1, lst = 0;
6       for(int i = 1; i <= m; i++){
7           dp[now][i] = dp[lst][i - 1] + a[i];
8           ll mx = dp[lst][i - 1];
9           for(int j = i + 1; j <= n - m + i; j++){
10              mx = max(mx, dp[lst][j - 1]);
11              dp[now][j] = max(dp[now][j - 1], mx) + a[j];
12          }
13          swap(now, lst);
14      }
15      ll ret = -linf;
16      for(int j = m; j <= n; ++j) ret = max(ret, dp[lst][j]);
17      return ret;
18  }
```

### 9.10 虚树 DP

只需按照需求修改 solve() 函数并用 lca() 维护相关信息即可。

给定一棵有 $n$ 个节点的树，$q$ 次询问，每次询问给 $k$ 个点，求至少删除多少点，使得这 $k$ 个点两两不属于同一联通块 $n, q, \sigma k \leq 100000$

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   typedef long long ll;
4   typedef unsigned long long ull;
5   const int inf = 0x3f3f3f3f;
6   const ll linf = 0x3f3f3f3f3f3f3f3f;
7   const ll mod = 998244353LL;
```

```
8
9  const int maxn = 1e5 + 5;
10 const int maxm = log2(maxn) + 2;
11 struct Graph {
12     struct Edge {
13         int to, nxt;
14     } E[maxn << 1];
15     int head[maxn], tot;
16     void init(int n) {
17         for (int i = 0; i <= n; ++i) head[i] = -1;
18         tot = 0;
19     }
20     void add_edge (int u, int v) {
21         E[tot] = {v, head[u]};
22         head[u] = tot++;
23     }
24 } G, VG;
25 int dfn[maxn], cnt;
26 int d[maxn], fa[maxn][maxm];
27 void dfs(int x, int y) {
28     dfn[x] = ++cnt;
29     for (int i = 1; i < maxm; ++i) {
30         fa[x][i] = fa[fa[x][i - 1]][i - 1];
31     }
32     for (int i = G.head[x]; i != -1; i = G.E[i].nxt) {
33         int to = G.E[i].to;
34         if (to == y) continue;
35         fa[to][0] = x;
36         d[to] = d[x] + 1;
37         dfs(to, x);
38     }
39 }
40 int lca(int a, int b) {
41     if (d[a] > d[b]) swap(a, b);
42     int dt = d[b] - d[a];
43     for (int i = 0; (1 << i) <= dt; ++i) {
44         if (((dt >> i) & 1) == 0) continue;
45         b = fa[b][i];
46     }
```

```
47      for (int i = maxm - 1; i >= 0; --i) {
48          if (fa[a][i] == fa[b][i]) continue;
49          a = fa[a][i]; b = fa[b][i];
50      }
51      if (a == b) return a;
52      return fa[a][0];
53  }
54  int id[maxn];
55  int st[maxn], tp;
56  int vis[maxn];
57  ll dp[maxn][2];
58  void solve(int x, int cq) {
59      ll sum0 = 0, sum1 = 0;
60      ll mx = -inf;
61      for (int i = VG.head[x]; i != -1; i = VG.E[i].nxt) {
62          int to = VG.E[i].to;
63          solve(to, cq);
64          if (d[to] > d[x] + 1) dp[to][0] = min(dp[to][0], dp[to][1] + 1LL);
65          sum0 += dp[to][0];
66          sum1 += dp[to][1];
67          mx = max(mx, dp[to][0] - dp[to][1]);
68      }
69      if (vis[x] == cq) {
70          dp[x][0] = inf;
71          dp[x][1] = sum0;
72      } else {
73          dp[x][0] = min(sum0, 1LL + sum1);
74          dp[x][1] = min(dp[x][0], sum0 - mx);
75      }
76      if (dp[x][0] > inf) dp[x][0] = inf;
77      if (dp[x][1] > inf) dp[x][1] = inf;
78  }
79  void vir_tree (int cq) {
80      int k; scanf("%d", &k);
81      for (int i = 0; i < k; ++i) {
82          scanf("%d", &id[i]);
83          vis[id[i]] = cq;
84      }
85      sort(id, id + k, [&](const int &x, const int &y) -> bool {
```

```
86          return dfn[x] < dfn[y];
87      });
88      tp = 0;
89      VG.head[1] = -1; st[tp++] = 1;
90      for (int i = 0; i < k; ++i) {
91          int x = id[i];
92          if (x == 1) continue;
93          int lx = lca(x, st[tp - 1]);
94          if (lx != st[tp - 1]) {
95              while (tp > 1 && dfn[lx] < dfn[st[tp - 2]]) {
96                  VG.add_edge(st[tp - 2], st[tp - 1]);
97                  --tp;
98              }
99              if (tp > 1 && dfn[lx] > dfn[st[tp - 2]]) {
100                 VG.head[lx] = -1;
101                 VG.add_edge(lx, st[tp - 1]);
102                 st[tp - 1] = lx;
103             } else {
104                 VG.add_edge(lx, st[tp - 1]);
105                 --tp;
106             }
107         }
108         VG.head[x] = -1; st[tp++] = x;
109     }
110     for (int i = 0; i < tp - 1; ++i) {
111         VG.add_edge(st[i], st[i + 1]);
112     }
113     solve(1, cq);
114     int ans = min(dp[1][0], dp[1][1]);
115     if (ans == inf) ans = -1;
116     printf("%d\n", ans);
117 }
118 int main() {
119     int n; scanf("%d", &n);
120     G.init(n);
121     VG.init(n);
122     for (int i = 1; i < n; ++i) {
123         int u, v; scanf("%d%d", &u, &v);
124         G.add_edge(u, v);
```

```
125         G.add_edge(v, u);
126     }
127     cnt = fa[1][0] = 0;
128     d[1] = 1;
129     dfs(1, -1);
130     for (int i = 1; i <= n; ++i) vis[i] = 0;
131     int q; scanf("%d", &q);
132     for (int cq = 1; cq <= q; ++cq) {
133         vir_tree(cq);
134     }
135     return 0;
136 }
```

# 10 博弈论

## 10.1 Bash 博弈

有一堆石子共有 N 个。A B 两个人轮流拿，A 先拿。每次最少拿 1 颗，最多拿 K 颗，拿到最后 1 颗石子的人获胜。假设 A B 都非常聪明，拿石子的过程中不会出现失误。给出 N 和 K，问最后谁能赢得比赛。

Bash 博弈先手必胜当且仅当 $N\%(K+1)=0$

## 10.2 Nim 博弈

有 N 堆石子。A B 两个人轮流拿，A 先拿。每次只能从一堆中取若干个，可将一堆全取走，但不可不取，拿到最后 1 颗石子的人获胜。假设 A B 都非常聪明，拿石子的过程中不会出现失误。给出 N 及每堆石子的数量，问最后谁能赢得比赛。

Nim 博弈先手必胜，当且仅当 $X_1 xor X_2 xor \ldots\ldots xor X_n \neq 0$

## 10.3 Wythoff 博弈

有 2 堆石子。A B 两个人轮流拿，A 先拿。每次可以从一堆中取任意个或从 2 堆中取相同数量的石子，但不可不取。拿到最后 1 颗石子的人获胜。假设 A B 都非常聪明，拿石子的过程中不会出现失误。给出 2 堆石子的数量，问最后谁能赢得比赛。

```
1 void Wythoff(int n, int m){
2     if(n > m) swap(n, m);
3     int tmp = (m - n)*(sqrt(5) + 1)/2;
4     if(n == tmp) puts("B");
5     else puts("A");
6 }
```

## 10.4 公平组合游戏

若一个游戏满足：

1. 游戏由两个人参与，二者轮流做出决策
2. 在游戏进程的任意时刻，可以执行的合法行动与轮到哪名玩家无关
3. 有一个人不能行动时游戏结束

则称这个游戏是一个公平组合游戏（Impartial Combinatorial Games，ICG）,NIM 游戏就是一个公平组合游戏

### 10.4.1 SG-组合游戏

一个公平组合游戏若满足：

1. 两人的决策都对自己最有利
2. 当有一人无法做出决策时游戏结束，无法做出决策的人输，且游戏一定能在有限步数内结束
3. 游戏中的同一个状态不可能多次抵达，且游戏不会出现平局

则这类游戏可以用 SG 函数解决，我们称之为 SG-组合游戏

### 10.4.2 Sprague-Grundy 函数

SG 函数是对游戏图中每一个节点的评估函数。

即一个状态的 SG 函数值等于 mex 其所有后继状态的 SG 函数值

其中，mex（minimal excludant）是定义在整数集合上的操作，它的自变量是任意整数集合，函数值是不属于该集合的最小自然数

f(X) == 0 当且仅当 X 为必败态

### 10.4.3 游戏的和

游戏的和是指：

考虑多个同时进行的 SG-组合游戏，这些 SG-组合游戏的和是这样的一个 SG-组合游戏，在它进行的过程中，游戏者可以任意挑选其中任意一个游戏进行决策

### 10.4.4 Sprague-Grundy 定理

游戏的和的 SG 函数值等于它包含的各个子游戏 SG 函数值得异或和。即

$$SG(G) = SG(G_1) xor SG(G_2) xor \cdots xor SG(G_m)$$

### 10.4.5 Anti-SG 游戏

定义：

Anti-SG 游戏规定，决策集合为空的游戏者赢。

Anti-SG 游戏的其他规则与 SG 游戏相同。

SG 定理：

对于任意一个 Anti-SG 游戏，如果我们规定当局面中所有的单一游戏的 SG 值为 0 时游戏结束，则先手必胜当且仅当：

（1）游戏的 SG 函数不为 0 且游戏中某个单一游戏的 SG 函数值大于 1 （2）游戏的 SG 函数为 0 且游戏中没有单一游戏的 SG 函数值大于 1

## 10.5 删边游戏

### 10.5.1 树的删边游戏

给出一个有 N 个点的树，有一个点作为树的根节点。

游戏者轮流从树中删边，删去一条边后，不与根节点相连的部分将被移走。

无法行动者输。

有如下定理：叶子节点的 SG 值为 0；其它节点的 SG 值为它的所有子节点的 SG 值加 1 后的异或和。

### 10.5.2 无向图删边游戏

一个无向连通图，有一个点作为图的根。

游戏者轮流从图中删去边，删去一条边后，不与根节点相连的部分被移走，无法行动者输。

Fusion Principle：

我们可以对无向图做如下改动：将图中的任意一个偶环缩成一个新点，任意一个奇环缩成一个新点加一个新边；所有连到原先环上的边全部改为与新点相连。这样的改动不会影响图的 SG 值。

这样我们就可以将任意一个无向图改成树结构。

# 11 字符串

## 11.1 最长回文子串（Manacher 算法）

```
1 const int maxn = 5e5 + 5;
2 int len;
3 char t[maxn << 1];
4 int p[maxn << 1];
5 char str[maxn];
6 void Manacher(){
7     int mx = 0, id = -1;
8     len = strlen(str);
```

```
9       for(int i = 0; i <= len*2; ++i) {
10          p[i] = 0;
11          t[i] = 0;
12      }
13      for(int i = 0; i <= len*2; ++i) {
14          if((i & 1) == 1) {
15              t[i] = str[i/2];
16          } else {
17              t[i] = '*';
18          }
19      }
20      for(int i = 1; i < len*2; ++i) {
21          if(mx > i) {
22              p[i] = min(p[2*id - i], mx - i);
23          } else {
24              p[i] = 1;
25          }
26          while(i - p[i] >= 0 && t[i + p[i]] == t[i - p[i]]) {
27              ++p[i];
28          }
29          if(p[i] + i > mx) {
30              mx = p[i] + i;
31              id = i;
32          }
33      }
34      for(int i = 0; i < len*2; ++i) {
35          int pos = i/2;
36          int plen = p[i] - 1;
37          int lf = pos - plen/2;
38          int rt = lf + plen - 1;
39          if(lf > rt) {
40              continue;
41          }
42          // str[lf, rt] is longest palindrome around str[pos]
43          printf("[%d, %d]\n", lf, rt);
44          for(int j = lf; j <= rt; ++j) {
45              putchar(str[j]);
46          }
47          puts("");
```

```
48      }
49  }
```

## 11.2 KMP 算法（单母串匹配）

### 11.2.1 模板

```
1  const int maxn = 1e6 + 10;
2  int nxt[maxn];
3  char s1[maxn], s2[maxn];
4  int len1, len2;
5  void getNext(){
6      int i = 0, j = -1;
7      nxt[0] = -1;
8      while(i < len2) {
9          if(j == -1 || s2[i] == s2[j]) nxt[++i] = ++j;
10         else j = nxt[j];
11     }
12 }
13
14 void kmp() {
15     int i = 0, j = 0;
16     getNext();
17     while(i < len1){
18         if(j == -1 || s1[i] == s2[j]) ++i, ++j;
19         else j = nxt[j];
20         // 匹配成功 s1[i - len2, i) = s2[0, len2)
21         if(j == len2) {
22 //              printf("%d\n", i - len2);
23             j = nxt[j];
24         }
25     }
26 }
```

### 11.2.2 循环节

如果对于 next 数组中的 i， 符合

$$i\%(i - next[i]) == 0 \&\& next[i]! = 0$$

则说明字符串循环，而且

循环节长度为： i - next[i]
循环次数为：i / ( i - next[i] )
对于一个不完整的循环串要补充多少个才能使得其完整

$$(\quad) - len\%(\quad)$$

即

$$(len - next[len]) - len\%(len - next[len])$$

### 11.2.3 拓展 kmp （求 T 与 S 的每一个后缀：s[i n - 1] 的最长公共前缀）

```
int nxt[maxn], ex[maxn];
void getNext(char *str) {
    int i = 0, j, pos, len = strlen(str);
    nxt[0] = len;
    while(str[i] == str[i + 1] && i + 1 < len) ++i;
    nxt[1] = i; pos = 1;
    for(i = 2; i < len; ++i){
        if(nxt[i - pos] + i < nxt[pos] + pos) {
            nxt[i] = nxt[i - pos];
            continue;
        }
        j = nxt[pos] + pos - i;
        if(j < 0) j = 0;
        while(i + j < len && str[j] == str[j + i]) ++j;
        nxt[i] = j; pos = i;
    }
}

void exkmp(char *s1, char *s2){
    int i = 0, j, pos, l1 = strlen(s1), l2 = strlen(s2);
    getNext(s2);
    while(s1[i] == s2[i] && i < l2 && i < l1) ++i;
    ex[0] = i; pos = 0;
    for(i = 1; i < l1; ++i){
        if(nxt[i - pos] + i < ex[pos] + pos) {
            ex[i] = nxt[i - pos];
            continue;
        }
        j = ex[pos] + pos - i;
```

```
30          if(j < 0) j = 0;
31          while(i + j < l1 && j < l2 && s1[j + i] == s2[j]) ++j;
32          ex[i] = j; pos = i;
33      }
34  }
```

## 11.3 AC 自动机（多母串匹配）

```
1   const int maxn = 1e6 + 5;
2   const int TrieSize = 1e6 + 5;
3   const int SigmaSize = 26;
4
5   namespace Trie {
6       int T[TrieSize][SigmaSize];
7       int num[TrieSize];
8       bool vis[TrieSize];
9       int tot;
10      void init() {
11          tot = 0;
12          num[0] = 0;
13          vis[0] = false;
14          for(int i = 0; i < SigmaSize; ++i) {
15              T[0][i] = 0;
16          }
17      }
18      int getID(char ch) {
19          int ret = ch - 'a';
20          return ret;
21      }
22      int newNode() {
23          ++tot;
24          num[tot] = 0;
25          vis[tot] = false;
26          for(int i = 0; i < SigmaSize; ++i) {
27              T[tot][i] = 0;
28          }
29          return tot;
30      }
31      void ist(char s[]) {
```

```
32          int len = strlen(s);
33          int now = 0;
34          for(int i = 0; i < len; ++i) {
35              int id = getID(s[i]);
36              if(T[now][id] == 0) {
37                  T[now][id] = newNode();
38              }
39              now = T[now][id];
40          }
41          ++num[now];
42      }
43  }
44  namespace ACAM {
45      using namespace Trie;
46      int F[TrieSize];
47      int lst[TrieSize];
48      void getFail() {
49          queue<int> q;
50          F[0] = lst[0] = 0;
51          for(int i = 0; i < SigmaSize; ++i) {
52              int to = T[0][i];
53              if(to != 0) {
54                  F[to] = lst[to] = 0;
55                  q.push(to);
56              }
57          }
58          while(!q.empty()) {
59              int now = q.front(); q.pop();
60              for(int i = 0; i < SigmaSize; ++i) {
61                  int to = T[now][i];
62                  if(to == 0) {
63                      T[now][i] = T[F[now]][i];
64                  } else {
65                      q.push(to);
66                      int fn = F[now];
67                      while(fn != 0 && T[fn][i] == 0) {
68                          fn = F[fn];
69                      }
70                      F[to] = T[fn][i];
```

```
71                      if(num[F[to]] > 0) {
72                          lst[to] = F[to];
73                      } else {
74                          lst[to] = lst[F[to]];
75                      }
76                  }
77              }
78          }
79      }
80      int calc(int now) {
81          int ret = 0;
82          while(now != 0 && !vis[now]) {
83              vis[now] = true;
84              ret += num[now];
85              now = lst[now];
86          }
87          return ret;
88      }
89      int query(char s[]) {
90          int len = strlen(s);
91          int ret = 0, now = 0;
92          for(int i = 0; i < len; ++i) {
93              int id = getID(s[i]);
94              now = T[now][id];
95              if(num[now] > 0) {
96                  ret += calc(now);
97              } else {
98                  int nxt = lst[now];
99                  if(nxt != 0) {
100                     ret += calc(nxt);
101                 }
102             }
103         }
104         return ret;
105     }
106     void show() {
107         puts("****************");
108         for(int i = 0; i <= tot; ++i) {
109             printf("Trie[%d]: \n", i);
```

```
110             printf("num: %d, fail: %d, last: %d\n", num[i], F[i], lst[i]);
111             printf("to:");
112             for(int j = 0; j < SigmaSize; ++j) {
113                 if(T[i][j] == 0) continue;
114                 printf(" (%c, %d)", j + 'a', T[i][j]);
115             }
116             puts("");
117         }
118         puts("****************");
119     }
120 }
121 char s[maxn];
122 int main() {
123     int T; scanf("%d", &T);
124     for(int cs = 1; cs <= T; ++cs) {
125         Trie::init();
126         int n; scanf("%d", &n);
127         for(int i = 0; i < n; ++i) {
128             scanf("%s", s);
129             Trie::ist(s);
130         }
131         // ACAM::show();
132         ACAM::getFail();
133         // ACAM::show();
134         scanf("%s", s);
135         printf("%d\n", ACAM::query(s));
136     }
137     return 0;
138 }
```

## 11.4 循环串的最小表示法

```
1  char s[maxn];
2  int getMin() {
3      int n = strlen(s);
4      int i = 0, j = 1, k = 0;
5      while(i < n && j < n && k < n){
6          int dt = s[(i + k)%n] - s[(j + k)%n];
7          if(dt == 0) ++k;
```

```
8           else{
9               if(dt > 0) i += k + 1;
10              else j += k + 1;
11              if(i == j) ++j;
12              k = 0;
13          }
14      }
15      // 返回字典序最小的循环串的起始位置
16      int ret = min(i, j);
17      return ret;
18  }
```

## 11.5 序列自动机

字符串下标需从 1 开始，会方便很多

nxt[i][j] 表示第 i 个字符的下一个 j 字母出现的位置 (在原串中的下标)

cnt[i][j] s 表示第 i 个字符后面 j 字母的个数

```
1   char s[maxn];
2   int nxt[maxn][27], pos[27];
3   int cnt[maxn][27], num[27];
4   void getSeqAM() {
5       int len = strlen(s + 1);
6       for(int j = 0; j < 26; ++j) {
7           pos[j] = 0;
8           num[j] = 0;
9       }
10      for(int i = len; i >= 1; --i) {
11          for(int j = 0; j < 26; ++j) {
12              nxt[i][j] = pos[j];
13              cnt[i][j] = num[j];
14          }
15          int id = s[i] - 'a';
16          pos[id] = i; ++num[id];
17      }
18      for(int j = 0; j < 26; ++j) {
19          nxt[0][j] = pos[j];
20          cnt[0][j] = num[j];
21      }
22  }
```

## 11.6 任意子串 Hash

字符串下标需从 1 开始

本质为倒序排列的 key 进制数

```
typedef unsigned long long ull; // %llu
const ull key = 137; // 233 769
const ull mod = 998244353; // 1000000007
ull H[maxn], xp[maxn];
void Init_Hash(char *s){
    int n = strlen(s + 1);
    H[0] = 0; xp[0] = 1;
    for(int i = 1; i <= n; ++i) H[i] = H[i - 1]*key + (s[i] - 'a');
    for(int i = 1; i <= n; ++i) xp[i] = xp[i - 1]*key;
}
ull Hash(int l, int r){
    return H[r] - H[l - 1] * xp[r - l + 1];
}
```

## 11.7 shift-and 算法

给你一个 $n$ 位数，每一位可以有 $m_i$ 种选择。

给定一个需要匹配的母串，让你输出母串中有多少个可行的 $n$ 位子串。

```
bitset<1005> b[10], ans;
char s[maxn];
int main() {
    int n; scanf("%d", &n);
    for(int i = 0; i < n; ++i) {
        int m; scanf("%d", &m);
        while(m--) {
            int x; scanf("%d", &x);
            b[x].set(i);
        }
    }
    scanf("%s", s);
    int len = strlen(s);
    for(int i = 0; i < len; ++i) {
        ans.set(0);
        ans &= b[s[i] - '0'];
        if(ans[n - 1] == 1) {
            // printf("[%d, %d]\n", i - n + 1, i);
```

```
19              char ch = s[i + 1];
20              s[i + 1] = '\0';
21              puts(s + i - n + 1);
22              s[i + 1] = ch;
23          }
24          ans <<= 1;
25      }
26      return 0;
27  }
```

## 11.8 后缀数组

后缀数组诱导排序 (SuffixArray-InducedSort)

```
1   const int maxn = 1e6 + 5;
2   namespace SA {
3       int s[maxn << 1], t[maxn << 1], p[maxn], cnt[maxn], cur[maxn];
4       int n, sa[maxn], rank[maxn], height[maxn], mi[maxn][25];
5       #define pushS(x) sa[cur[s[x]]--] = x
6       #define pushL(x) sa[cur[s[x]]++] = x
7       #define inducedSort(v) fill_n(sa, n, -1); fill_n(cnt, m, 0); \
8           for(int i = 0; i < n; ++i) cnt[s[i]]++; \
9           for(int i = 1; i < m; ++i) cnt[i] += cnt[i-1]; \
10          for(int i = 0; i < m; ++i) cur[i] = cnt[i]-1; \
11          for(int i = n1 - 1; i != -1; --i) pushS(v[i]); \
12          for(int i = 1; i < m; ++i) cur[i] = cnt[i-1]; \
13          for(int i = 0; i < n; ++i) if(sa[i] > 0 && t[sa[i] - 1]) pushL(sa
                [i] - 1); \
14          for(int i = 0; i < m; ++i) cur[i] = cnt[i]-1; \
15          for(int i = n - 1; i != -1; --i) if(sa[i] > 0 && !t[sa[i]-1])
                pushS(sa[i]-1)
16      void sais(int n, int m, int *s, int *t, int *p) {
17          int n1 = t[n - 1] = 0, ch = rank[0] = -1, *s1 = s + n;
18          for(int i = n - 2; i != -1; --i) t[i] = s[i] == s[i + 1] ? t[i +
                1] : s[i] > s[i + 1];
19          for(int i = 1; i < n; ++i) rank[i] = (t[i-1] && !t[i]) ? (p[n1] =
                i, n1++) : -1;
20          inducedSort(p);
21          for(int i = 0, x, y; i < n; ++i) if(~(x = rank[sa[i]])) {
22              if(ch < 1 || p[x+1] - p[x] != p[y+1] - p[y]) ++ch;
```

```
23              else for(int j = p[x], k = p[y]; j <= p[x + 1]; ++j, ++k)
24                  if((s[j] << 1 | t[j]) != (s[k] << 1 | t[k])) {++ch; break
                        ;}
25              s1[y = x] = ch;
26          }
27          if(ch + 1 < n1) sais(n1, ch + 1, s1, t + n, p + n1);
28          else for(int i = 0; i < n1; i++) sa[s1[i]] = i;
29          for(int i = 0; i < n1; i++) s1[i] = p[sa[i]];
30          inducedSort(s1);
31      }
32      template<typename T>
33      int mapCharToInt(int n, const T *str) {
34          int m = *max_element(str, str + n);
35          fill_n(rank, m + 1, 0);
36          for(int i = 0; i < n; i++) rank[str[i]] = 1;
37          for(int i = 0; i < m; i++) rank[i + 1] += rank[i];
38          for(int i = 0; i < n; i++) s[i] = rank[str[i]] - 1;
39          return rank[m];
40      }
41      template<typename T>
42      void suffixArray(int n, const T *str) {
43          int m = mapCharToInt(++n, str);
44          sais(n, m, s, t, p);
45          for(int i = 0; i < n; i++) rank[sa[i]] = i;
46          for(int i = 0, h = height[0] = 0; i < n - 1; ++i) {
47              int j = sa[rank[i] - 1];
48              while(i + h < n && j + h < n && s[i + h] == s[j + h]) ++h;
49              if(height[rank[i]] = h) --h;
50          }
51      }
52      void RMQ_init() {
53          for(int i = 0; i < n; ++i) mi[i][0] = height[i + 1];
54          for(int j = 1; (1 << j) <= n; ++j){
55              for(int i = 0; i + (1 << j) <= n; ++i){
56                  mi[i][j] = min(mi[i][j - 1], mi[i + (1 << (j - 1))][j -
                        1]);
57              }
58          }
59      }
```

```
60      int RMQ(int L, int R) {
61          int k = 0, len = R - L + 1;
62          while((1 << (k + 1)) <= len) ++k;
63          return min(mi[L][k], mi[R - (1 << k) + 1][k]);
64      }
65      int LCP(int i, int j) {
66          if(rank[i] > rank[j]) swap(i, j);
67          return RMQ(rank[i], rank[j] - 1);
68      }
69      template<typename T>
70      void init(T *str){
71          n = strlen(str);
72          str[n] = 0;
73          suffixArray(n, str);
74          RMQ_init();
75      }
76      void print() {
77          puts("Suffix Array Print Start");
78          for(int i = 0; i <= n; ++i) {
79              printf("s[%d] = %d, sa = %d rank = %d height = %d\n", i, s[i],
                    sa[i], rank[i], height[i]);
80          }
81          puts("Suffix Array Print End");
82      }
83  };
```

## 11.9 后缀自动机（将所有子串压缩到一个 DAG 中）

### 11.9.1 模板

```
1   const int maxn = 1e6 + 5;
2   const int MaxState = 2*maxn; // 至少开字符串长度的两倍
3   const int SigmaSize = 26;
4   template<class T> class SAM{
5   private:
6       int tot, lst, len;
7       int trans[MaxState][SigmaSize], mx[MaxState], mi[MaxState], link[
            MaxState];
8       int newState() {
9           const int x = tot++;
```

```
10          for(int i = 0; i < SigmaSize; ++i) {
11              trans[x][i] = -1;
12          }
13          link[x] = -1;
14          mx[x] = mi[x] = 0;
15          return x;
16      }
17      void init() {
18          tot = 0;
19          lst = newState();
20      }
21      int getId(T x) {
22          return (int)(x - 'a');
23          // return (int)(x - 1);
24      }
25      void add(int ch) {
26          int p = lst;
27          int np = lst = newState();
28          mx[np] = mx[p] + 1;
29          while(p != -1 && trans[p][ch] == -1) {
30              trans[p][ch] = np;
31              p = link[p];
32          }
33          if(p == -1) {
34              mi[np] = 1;
35              link[np] = 0;
36              return ;
37          }
38          int q = trans[p][ch];
39          if(mx[p] + 1 == mx[q]){
40              mi[np] = mx[q] + 1;
41              link[np] = q;
42              return ;
43          }
44          int nq = newState();
45          mx[nq] = mx[p] + 1;
46          link[nq] = link[q];
47          for(int i = 0; i < SigmaSize; ++i) {
48              trans[nq][i] = trans[q][i];
```

```
49          }
50          mi[np] = mi[q] = mx[nq] + 1;
51          link[np] = link[q] = nq;
52          while(p != -1 && trans[p][ch] == q) {
53              trans[p][ch] = nq;
54              p = link[p];
55          }
56          mi[nq] = mx[link[nq]] + 1;
57      }
58      int tax[maxn], topo[MaxState]; // Topology order: topo[0, tot - 1]
59      void getTopo() {
60          for(int i = 0; i <= len; ++i) tax[i] = 0;
61          for(int i = 0; i < tot; ++i) ++tax[mx[i]];
62          for(int i = 1; i <= len; ++i) tax[i] += tax[i - 1];
63          for(int i = 0; i < tot; ++i) topo[--tax[mx[i]]] = i;
64      }
65      void show() {
66          for(int i = 0; i < tot; ++i) {
67              printf("State %d: \n", i);
68              printf("transition:");
69              for(int j = 0; j < SigmaSize; ++j) {
70                  printf(" %d", trans[i][j]);
71              }
72              puts("");
73              printf("link: %d, min length: %d, max length: %d\n", link[i],
                    mi[i], mx[i]);
74              puts("++++++++++++++++++++");
75          }
76          printf("Topology order:");
77          for(int i = 0; i < tot; ++i) {
78              printf(" %d", topo[i]);
79          }
80          puts("");
81      }
82  public:
83      SAM() {
84          tot = 0;
85      }
86      int id[maxn]; // always need
```

```
87      void solve(int n, T a[]) {
88          len = n;
89          init();
90          for(int i = 0; i < len; ++i) {
91              id[i] = getId(a[i]);
92              add(id[i]);
93          }
94          // 此行以下根据题目要求进行修改
95          getTopo();
96          // show();
97
98      }
99  };
```

调用方式

```
1  SAM<char> sam;
2  char a[maxn];
3  int main() {
4      scanf("%s", a);
5      int n = strlen(a);
6      sam.solve(n, a);
7      return 0;
8  }
```

### 11.9.2 字符串本质不同的非空子串个数

```
1  void solve(int n, T a[]) {
2      len = n;
3      init();
4      for(int i = 0; i < len; ++i) {
5          id[i] = getId(a[i]);
6          add(id[i]);
7      }
8      ll ans = 0;
9      for(int i = 1; i < tot; ++i) {
10         ans += (ll)(mx[i] - mi[i] + 1);
11     }
12     printf("%lld\n", ans);
13 }
```

### 11.9.3 字符串的所有长度为 K 的子串出现次数最多的子串的出现次数

```
int cnt[MaxState], num[maxn], ans[maxn];
void solve(int n, T a[]) {
    len = n;
    init();
    for(int i = 0; i < len; ++i) {
        id[i] = getId(a[i]);
        add(id[i]);
    }
    // 此行以下根据题目要求进行修改
    getTopo();
    // show();
    for(int i = 0; i < tot; ++i) cnt[i] = 0;
    int p = 0;
    for(int i = 0; i < len; ++i) {
        p = trans[p][id[i]];
        ++cnt[p];
    }
    for(int i = tot - 1; i >= 0; --i) {
        int x = topo[i];
        if(link[x] != -1) {
            cnt[link[x]] += cnt[x];
        }
    }
    for(int i = 0; i <= len; ++i) {
        num[i] = 0;
    }
    for(int i = 0; i < tot; ++i) {
        num[mx[i]] = max(num[mx[i]], cnt[i]);
    }
    int val = 0;
    for(int i = len; i >= 1; --i) {
        val = max(val, num[i]);
        ans[i] = val;
    }
    for(int i = 1; i <= len; ++i) {
        printf("%d\n", ans[i]);
    }
}
```

### 11.9.4 最长不重叠子串

```
int lf[MaxState], rt[MaxState]; // 该状态（的结尾位置）出现的最左位置与最右位置 max{endpos}, min{endpos}
void solve(int n, T a[]) {
    len = n;
    init();
    for(int i = 0; i < len; ++i) {
        id[i] = getId(a[i]);
        add(id[i]);
    }
    // 此行以下根据题目要求进行修改
    getTopo();
    // show();
    for(int i = 0; i < tot; ++i) {
        lf[i] = inf;
        rt[i] = -inf;
    }
    int p = 0;
    for(int i = 0; i < len; ++i) {
        p = trans[p][id[i]];
        lf[p] = min(lf[p], i);
        rt[p] = max(rt[p], i);
    }
    for(int i = tot - 1; i >= 0; --i) {
        int x = topo[i];
        int lx = link[x];
        if(lx != -1) {
            lf[lx] = min(lf[lx], lf[x]);
            rt[lx] = max(rt[lx], rt[x]);
        }
    }
    int ans = 0;
    for(int i = 1; i < tot; ++i) {
        // printf("%d: %d [%d, %d]\n", i, mx[i], lf[i], rt[i]);
        int now = min(mx[i], rt[i] - lf[i]);
        ans = max(ans, now);
    }
    printf("%d\n", ans);
}
```

### 11.9.5 字典序第 K 小子串

对于一个给定长度为 N 的字符串，求它的第 K 小子串。

T 为 0 则表示不同位置的相同子串算作一个。T 为 1 则表示不同位置的相同子串算作多个。

```
ll cnt[MaxState], sum[MaxState];
void dfs(int p, int k) {
    k -= cnt[p];
    if(k <= 0) {
        return ;
    }
    for(int i = 0; i < SigmaSize; ++i) {
        int to = trans[p][i];
        if(to == -1) {
            continue;
        }
        if(sum[to] >= k) {
            printf("%c", 'a' + i);
            dfs(to, k);
            return ;
        }
        k -= sum[to];
    }
}
void solve(int n, T a[]) {
    len = n;
    init();
    for(int i = 0; i < len; ++i) {
        id[i] = getId(a[i]);
        add(id[i]);
    }
    // 此行以下根据题目要求进行修改
    getTopo();
    // show();
    int t, k; scanf("%d%d", &t, &k);
    for(int i = 0; i < tot; ++i) {
        cnt[i] = 0;
    }
    if(t == 0) {
        for(int i = 0; i < tot; ++i) {
            cnt[i] = 1;
```

```
        }
    } else {
        int p = 0;
        for(int i = 0; i < len; ++i) {
            p = trans[p][id[i]];
            ++cnt[p];
        }
        for(int i = tot - 1; i > 0; --i) {
            int x = topo[i];
            int lx = link[x];
            cnt[lx] += cnt[x];
        }
    }
    cnt[0] = 0;
    for(int i = 0; i < tot; ++i) {
        sum[i] = cnt[i];
    }
    for(int i = tot - 1; i >= 0; --i) {
        int x = topo[i];
        for(int j = 0; j < SigmaSize; ++j) {
            int to = trans[x][j];
            if(to != -1) {
                sum[x] += sum[to];
            }
        }
    }
    if(sum[0] < k) {
        printf("-1");
    } else {
        dfs(0, k);
    }
    puts("");
}
```

### 11.9.6 多个字符串的最长公共子串

以第一个字符串建 SAM，其余字符串在 SAM 上进行转移并维护相关信息

```
char t[maxn];
int ans[MaxState], num[MaxState];
```

```
3  void solve(int n, T a[]) {
4      len = n;
5      init();
6      for(int i = 0; i < len; ++i) {
7          id[i] = getId(a[i]);
8          add(id[i]);
9      }
10     // 此行以下根据题目要求进行修改
11     getTopo();
12     // show();
13     for(int i = 0; i < tot; ++i) {
14         ans[i] = mx[i];
15     }
16     while(scanf("%s", t) != EOF) {
17         // if(t[0] == 'Q') break;
18         int lt = strlen(t);
19         for(int i = 0; i < tot; ++i) {
20             num[i] = 0;
21         }
22         int p = 0, now = 0;
23         for(int i = 0; i < lt; ++i) {
24             int cid = getId(t[i]);
25             if(trans[p][cid] == -1) {
26                 while(p != -1 && trans[p][cid] == -1) p = link[p];
27                 if(p == -1) {
28                     p = 0;
29                     now = 0;
30                 } else {
31                     now = mx[p] + 1;
32                     p = trans[p][cid];
33                 }
34             } else {
35                 ++now;
36                 p = trans[p][cid];
37             }
38             num[p] = max(num[p], now);
39         }
40         for(int i = tot - 1; i >= 1; --i) {
41             int x = topo[i];
```

```
            int to = link[x];
            num[to] = max(num[to], num[x]);
            if(num[to] > mx[to]) num[to] = mx[to];
        }
        for(int i = 1; i < tot; ++i) ans[i] = min(ans[i], num[i]);
    }
    int lcs = 0;
    for(int i = 1; i < tot; ++i) lcs = max(lcs, ans[i]);
    printf("%d\n", lcs);
}
```

## 11.10 Lyndon 分解

Lyndon 串：对于字符串 $s$，如果 $s$ 的字典序严格小于 $s$ 的所有后缀的字典序，称是简单串，或者 Lyndon 串。举一些例子，a , b , ab , aab , abb , ababb , abcd 都是 Lyndon 串。当且仅当的字典序严格小于它的所有非平凡的循环同构串时，才是 Lyndon 串。

Lyndon 分解：串 $s$ 的 Lyndon 分解记为 $s = w_1 w_2 \dots w_k$ ，其中所有 $w_i$ 为简单串，并且他们的字典序按照非严格单减排序，即 $w_1 \geq w_2 \geq \cdots \geq w_k$ 。这样的分解存在且唯一。

```
vector<string> duval(string const& s) {
    int n = s.size(), i = 0;
    vector<string> factorization;
    while (i < n) {
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j]) k = i;
            else k++;
            j++;
        }
        while (i <= k) {
            factorization.push_back(s.substr(i, j - k));
            i += j - k;
        }
    }
    return factorization;
}
```

## 11.11 回文自动机

```
struct PAM {
```

```
2     static const int N = 1e5 + 5;
3     int sz, tot, last;
4     struct Node {
5         int ch[26], len, fail;
6         int cnt, dep, dif, slink;
7         Node(int l = 0): len(l) {}
8     } node[N];
9     char s[N];
10    int newNode(int l) {
11        node[sz] = Node(l);
12        return sz++;
13    }
14    void clear() {
15        sz = -1, last = 0, tot = 0;
16        s[0] = '$';
17        newNode(0);
18        newNode(-1);
19        node[0].fail = 1;
20    }
21    int getfail(int x) {
22        while(s[tot - node[x].len - 1] != s[tot]) x = node[x].fail;
23        return x;
24    }
25    void insert(char c) {
26        s[++tot] = c;
27        int code = c - 'a';
28        Node& now = node[getfail(last)];
29        if (!now.ch[code]) {
30            int itx = newNode(now.len + 2);
31            Node &x = node[itx];
32            x.fail = node[getfail(now.fail)].ch[code];
33            Node &fx = node[x.fail];
34            x.dep = fx.dep + 1;
35            now.ch[code] = itx;
36            x.dif = x.len - fx.len;
37            if (x.dif == fx.dif)
38                x.slink = fx.slink;
39            else
40                x.slink = x.fail;
```

```
41          }
42          last = now.ch[code];
43          node[last].cnt++;
44      }
45  };
```

# 12 杂项

## 12.1 java 输入输出挂

```
1  static class InputReader {
2      public BufferedReader reader;
3      public StringTokenizer tokenizer;
4      public InputReader(InputStream stream) {
5          reader = new BufferedReader(new InputStreamReader(stream), 32768);
6          tokenizer = null;
7      }
8      public String next() {
9          while(tokenizer == null || !tokenizer.hasMoreTokens()) {
10             try {
11                 tokenizer = new StringTokenizer(reader.readLine());
12             } catch (IOException e) {
13                 throw new RuntimeException(e);
14             }
15         }
16         return tokenizer.nextToken();
17     }
18     public int nextInt() {
19         return Integer.parseInt(next());
20     }
21     public long nextLong() {
22         return Long.parseLong(next());
23     }
24     public double nextDouble() {
25         return Double.parseDouble(next());
26     }
27 }
```

## 12.2 模拟退火（Simulate Anneal）

Sta：状态结构体

J(y)：在状态 y 时的评价函数值

r：用于控制降温的快慢

T：系统的温度，系统初始应该要处于一个高温的状态

$T_{min}$：温度的下限，若温度 $T$ 达到 $T_{min}$，则停止搜索

```
1     const double T_min = 1e-3;
2  const double r = 0.98;
3  struct Sta{
4      int x, y;
5      Sta(int x = 0, int y = 0):x(x), y(y){}
6  };
7  Sta move(Sta now){
8      Sta ret;
9      return ret;
10 }
11 int J(Sta x){
12     int ret = 0;
13
14     return ret;
15 }
16 int simulate_anneal(double T){
17     srand(time(NULL));
18     int ret = 0;
19     Sta now = Sta();
20     while(T > T_min){
21         Sta nxt = move(now);
22         int j1 = J(now), j2 = J(nxt);
23         int dE = j2 - j1;
24         if(ret < j1) ret = j1;
25         if(dE >= 0){ //表达移动后得到更优解，则总是接受移动
26             now = nxt; //接受从Y(i)到Y(i+1)的移动
27         }else{
28             // 函数exp(dE/T)的取值范围是(0, 1)，dE/T越大，则exp(dE/T)也
29             double val = exp(dE/T), rn = (double)(rand()%123456)/123456.0;
30             if(val > rn)
31                 now = nxt; //接受从Y(i)到Y(i+1)的移动
32         }
33         T = r*T ; //降温退火 ，0<r<1 。r越大，降温越慢；r越小，降温越快
```

```
34          /*
35          * 若r过大，则搜索到全局最优解的可能会较高，但搜索的过程也就较长。
                若r过小，则搜索的过程会很快，但最终可能会达到一个局部最优值
36          */
37      }
38      return ret;
39  }
```

## 12.3 慢速乘（防止乘法溢出）

```
1   ll mulmod(ll a, ll b){
2       ll ans = 0;
3       for(; b; b>>=1){
4           if(b & 1) ans = (ans + a) % mod;
5           a = a * 2 % mod;
6       }
7       return ans;
8   }
```

## 12.4 三分查找

```
1   const ll linf = 0x3f3f3f3f3f3f3f3f;
2   ll cal(int x){
3       ll ret = 0;
4
5       return ret;
6   }
7   ll solve(int lf, int rt){
8       while(lf + 2 < rt){
9           int len = rt - lf;
10          int mid1 = len/3 + lf;
11          int mid2 = len/3*2 + lf;
12          // 此处为三分查找最小值
13          if(cal(mid1) >= cal(mid2)) lf = mid1;
14          else rt = mid2;
15      }
16      ll ret = linf;
17      for(int i = lf; i <= rt; ++i) ret = min(ret, cal(i));
18      return ret;
```

```
}
```

## 12.5 分数类

```
template <typename T>
struct Frac {
    T up, down;
    Frac(T u = 0, T d = 1) {
        T g = __gcd(u, d);
        up = u/g; down = d/g;
        if(up > 0 && down < 0) {
            up = -up;
            down = -down;
        }
    }
    bool operator <(const Frac& f) const {
        if(up*f.down < down*f.up) return true;
        return false;
    }
    bool operator ==(const Frac& f) const {
        if(up == f.up && down == f.down) return true;
        return false;
    }
    Frac operator +(const Frac& f) const {
        T g = __gcd(down, f.down);
        T u = f.down/g*up + down/g*f.up;
        T d = down/g*f.down;
        return Frac(u, d);
    }
    Frac operator -(const Frac& f) const {
        T g = __gcd(down, f.down);
        T u = f.down/g*up - down/g*f.up;
        T d = down/g*f.down;
        return Frac(u, d);
    }
    Frac operator *(const Frac& f) const {
        T g1 = __gcd(up, f.down);
        T g2 = __gcd(f.up, down);
        T u = (up/g1)*(f.up/g2);
```

```
36          T d = (f.down/g1)*(down/g2);
37          return Frac(u, d);
38      }
39      Frac operator /(const Frac& f) const {
40          T g1 = __gcd(up, f.up);
41          T g2 = __gcd(down, f.down);
42          T u = (up/g1)*(f.down/g2);
43          T d = (f.up/g1)*(down/g2);
44          return Frac(u, d);
45      }
46      void pt() {
47          cout << up;
48          // 根据需要修改
49          if(down != 1) cout << "/" << down;
50          cout << endl;
51      }
52  };
53  Frac<ll> num;
```

## 12.6 十进制水仙花数

```
1       // [0, 88] 十进制自然数中共 89 个水仙花数
2   string str[90] = {
3       "0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
4       "153", "370", "371", "407",
5       "1634", "8208", "9474",
6       "54748", "92727", "93084",
7       "548834",
8       "1741725", "4210818", "9800817", "9926315",
9       "24678050", "24678051", "88593477",
10      "146511208", "472335975", "534494836", "912985153",
11      "4679307774",
12      "32164049650", "32164049651", "40028394225", "42678290603", "
            44708635679", "49388550606", "82693916578", "94204591914",
13      "28116440335967",
14      "4338281769391370", "4338281769391371",
15      "21897142587612075", "35641594208964132", "35875699062250035",
16      "1517841543307505039", "3289582984443187032", "4498128791164624869", "
            4929273885928088826",
```

```
    "63105425988599693916",
    "128468643043731391252", "449177399146038697307",
    "2188769684112291628885 8", "27879694893054074471405", "
       279078650099770525678 14", "28361281321319229463398", "
       354525901040316919359 43",
    "1740880059380652930237 22", "1884514854478978960368 75", "
       239313664430041569350 093",
    "1550475334214501539088894", "1553242162893771850669378", "
       370690799595547598864 4380", "3706907995955475988644381", "
       442209511809589961945 7938",
    "1212049985636133724054 38066", "1212706960068013143284393 76", "
       128851796696487777842012787", "1746504644995313776316392 54", "
       177265453171792792366489765",
    "1460764061297198037261487 3089", "1900817413625427999501273 4740", "
       190081741362542799950127 34741", "2386671643552397598039036 9295",
    "1145037275765491025924292050346", "1927890457142960697580636236639",
       "230909268261619030750969533 8915",
    "1733350999778224930872510396 2772",
    "1867099610015387901006341329 76990", "
       186709961001538790100634132976991",
    "1122763285329372541592822900204593",
    "1263936951710379032894780720147 8392", "
       126799377802722785663038855941 96922",
    "1219167219625434121569735803609966019",
    "1281579207836605995509977054529 6129367",
    "1151322190187639925650955979739715 22400", "
       115132219018763992565095597973971522401"
};
```

## 12.7 牛顿迭代法（大数开方）

```
import java.util.*;
import java.math.BigInteger;
public class Main {
    public static void main(String[] args) {
        Scanner cin = new Scanner(System.in);
        BigInteger n = cin.nextBigInteger();
        int len = n.toString().length()/2;
        BigInteger x0 = new BigInteger(n.toString().substring(len));
```

```
9           while(true) {
10              BigInteger x1 = x0.multiply(x0).add(n).divide(BigInteger.
                    valueOf(2).multiply(x0));
11              if(x1.compareTo(x0) == 0) break;
12              x0 = x1;
13          }
14          System.out.println(x0);
15          cin.close();
16      }
17  }
```

## 12.8 一些奇奇怪怪的性质

- 给定母串，多次询问子串，保证子串总长度小于等于 sum，则有，其询问的字符串长度种类数最多为 $O(\sqrt{sum})$
- 若顶点坐标为整数，坐标值的范围不超过 M 的凸多边形的顶点数只有 $O((sqrt(M))^{\frac{2}{3}})$ $O(M^{\frac{1}{3}})$ ?)
- 字典序最小的拓扑序：拓扑排序 + 优先队列
- 最小值最早出现（次小值在最小值最早出现的前提下最早出现.....）的拓扑序：反向建图，跑出字典序最大的拓扑排序，倒序输出
- 随机边权的图，最短路径上的边数很少，生成树不唯一的概率为 0

## 12.9 丢人

- 调用 rand 函数之前必须使用 srand 设置随机种子
- 二分输出方案时, 最后要再 judge 一次正解
- 对于负数 /2 向 0 取整, » 向下取整
- 二叉树 (线段树, 堆...) 中注意儿子是 «, 父亲是 »