JB/ICPC Template Manaual

Northeast Agricultural University

Carry Rainyday

November 20, 2019

Contents

0 Header

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

#define fastin                      \
    ios_base::sync_with_stdio(0); \
    cin.tie(0);
typedef long long ll;
typedef long double ld;
typedef pair<int, int> PII;
typedef vector<int> VI;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
const double eps = 1e-6;

#ifndef ONLINE_JUDGE
#define dbg(args...)                              \
    do                                            \
    {                                             \
        cout << "\033[32;1m" << #args << " -> "; \
        err(args);                                \
    } while (0)
#else
#define dbg(...)
#endif
void err()
{
    cout << "\033[39;0m" << endl;
}
template <template <typename...> class T, typename t, typename... Args>
void err(T<t> a, Args... args)
{
    for (auto x : a) cout << x << ' ';
    err(args...);
}
template <typename T, typename... Args>
void err(T a, Args... args)
{
    cout << a << ' ';
    err(args...);
}

int main()
{
#ifndef ONLINE_JUDGE
    freopen("test.in", "r", stdin);
    freopen("test.out", "w", stdout);
#endif

    return 0;
}
```

1 Math

1.1 Prime

1.1.1 Eratosthenes Sieve

$O(n \log \log n)$ 筛出 maxn 内所有素数
$notprime[i] = 0/1$ 0 为素数 1 为非素数

```
const int maxn = "Edit";
bool notprime[maxn] = {1, 1};   // 0 && 1 为非素数
void GetPrime()
{
    for (int i = 2; i < maxn; i++)
        if (!notprime[i] && i <= maxn / i)  // 筛到√n为止
            for (int j = i * i; j < maxn; j += i)
                notprime[j] = 1;
}
```

1.1.2 Eular Sieve

$O(n)$ 得到欧拉函数 $phi[]$、素数表 $prime[]$、素数个数 tot

```
const int maxn = "Edit";
bool vis[maxn];
int tot, phi[maxn], prime[maxn];
void CalPhi()
{
    phi[1] = 1;
    for (int i = 2; i < maxn; i++)
    {
        if (!vis[i])
            prime[tot++] = i, phi[i] = i - 1;
        for (int j = 0; j < tot; j++)
        {
            if (i * prime[j] > maxn) break;
            vis[i * prime[j]] = 1;
            if (i % prime[j] == 0)
            {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            else
                phi[i * prime[j]] = phi[i] * (prime[j] - 1);
        }
    }
}
```

$d(n)$ 函数

```
const int maxn = "Edit";
int prime[maxn], tot;
int d[maxn], e[maxn]; //d正除数个数，e最小质因子个数
bool check[maxn];
void CalD()
{
    d[1] = 1;
```

```
    for (int i = 2; i < maxn; i++)
    {
        if (!check[i])
        {
            prime[tot++] = i;
            e[i] = 1, d[i] = 2;
        }
        for (int j = 0; j < tot; j++)
        {
            if (i * prime[j] >= maxn) break;
            check[i * prime[j]] = true;
            if (i % prime[j] == 0)
            {
                e[i * prime[j]] = e[i] + 1;
                d[i * prime[j]] = d[i] / e[i] * (e[i] + 1);
                break;
            }
            else
            {
                e[i * prime[j]] = 1;
                d[i * prime[j]] = 2 * d[i];
            }
        }
    }
}
```

$\sigma\lambda(n)$ 函数，$\lambda = 1$

```
const int maxn = "Edit";
int prime[maxn], tot;
int sig[maxn], e[maxn]; //sig正除数，e不含能整除i的最小质因子的正除数和
bool check[maxn];
void CalSig()
{
    sig[1] = 1;
    for (int i = 2; i < maxn; i++)
    {
        if (!check[i])
        {
            prime[tot++] = i;
            e[i] = 1, sig[i] = i + 1;
        }
        for (int j = 0; j < tot; j++)
        {
            if (i * prime[j] >= maxn) break;
            check[i * prime[j]] = true;
            if (i % prime[j] == 0)
            {
                sig[i * prime[j]] = sig[i] * prime[j] + e[i];
                e[i * prime[j]] = e[i];
                break;
            }
            else
            {
                sig[i * prime[j]] = sig[i] * (prime[j] + 1);
                e[i * prime[j]] = sig[i];
            }
        }
    }
}
```

1.1.3 Prime Factorization

```
vector<pair<ll, int>> getFactors(ll x)
{
    vector<pair<ll, int>> fact;
    for (int i = 0; prime[i] <= x / prime[i]; i++)
    {
        if (x % prime[i] == 0)
        {
            fact.emplace_back(prime[i], 0);
            while (x % prime[i] == 0) fact.back().second++, x /= prime[i];
        }
    }
    if (x != 1) fact.emplace_back(x, 1);
    return fact;
}
```

1.1.4 Miller Rabin

$O(s\log n)$ 内判定 2^{63} 内的数是不是素数，s 为测定次数

```
bool Miller_Rabin(ll n, int s)
{
    if (n == 2) return 1;
    if (n < 2 || !(n & 1)) return 0;
    int t = 0;
    ll  x, y, u = n - 1;
    while ((u & 1) == 0) t++, u >>= 1;
    for (int i = 0; i < s; i++)
    {
        ll a = rand() % (n - 1) + 1;
        ll x = Pow(a, u, n);
        for (int j = 0; j < t; j++)
        {
            ll y = Mul(x, x, n);
            if (y == 1 && x != 1 && x != n - 1) return 0;
            x = y;
        }
        if (x != 1) return 0;
    }
    return 1;
}
```

1.1.5 Segment Sieve

对区间 $[a,b)$ 内的整数执行筛法。
函数返回区间内素数个数
is_prime[i-a]=true 表示 i 是素数
$1<a<b\le 10^{12}, b-a\le 10^6$

```
const int maxn = "Edit";
bool is_prime_small[maxn], is_prime[maxn];
ll prime[maxn];
int segment_sieve(ll a, ll b)
{
    int tot = 0;
    for (ll i = 0; i * i < b; ++i) is_prime_small[i] = true;
```

```
    for (ll i = 0; i < b - a; ++i) is_prime[i] = true;
    for (ll i = 2; i * i < b; ++i)
        if (is_prime_small[i])
        {
            for (ll j = 2 * i; j * j < b; j += i)
                is_prime_small[j] = false;
            for (ll j = max(2LL, (a + i - 1) / i) * i; j < b; j += i)
                is_prime[j - a] = false;
        }
    for (ll i = 0; i < b - a; ++i)
        if (is_prime[i]) prime[tot++] = i + a;
    return tot;
}
```

1.2 Euler phi

1.2.1 Euler

```
ll euler(ll n)
{
    ll rt = n;
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0)
        {
            rt -= rt / i;
            while (n % i == 0) n /= i;
        }
    if (n > 1) rt -= rt / n;
    return rt;
}
```

1.2.2 Sieve

```
const int N = "Edit";
int phi[N] = {0, 1};
void caleuler()
{
    for (int i = 2; i < N; i++)
        if (!phi[i])
            for (int j = i; j < N; j += i)
            {
                if (!phi[j]) phi[j] = j;
                phi[j] = phi[j] / i * (i - 1);
            }
}
```

1.3 Basic Number Theory

1.3.1 Extended Euclidean

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

1.3.2 ax+by=c

引用返回通解: $X = x + k * dx, Y = y - k * dy$
引用返回的 x 是最小非负整数解，方程无解函数返回 0

```
#define Mod(a, b) (((a) % (b) + (b)) % (b))
bool solve(ll a, ll b, ll c, ll& x, ll& y, ll& dx, ll& dy)
{
    if (a == 0 && b == 0) return 0;
    ll x0, y0;
    ll d = exgcd(a, b, x0, y0);
    if (c % d != 0) return 0;
    dx = b / d, dy = a / d;
    x = Mod(x0 * c / d, dx);
    y = (c - a * x) / b;
    //  y = Mod(y0 * c / d, dy); x = (c - b * y) / a;
    return 1;
}
```

1.3.3 Multiplicative Inverse Modulo

利用 exgcd 求 a 在模 m 下的逆元，需要保证 $\gcd(a, m) == 1$.

```
ll inv(ll a, ll m)
{
    ll x, y;
    ll d = exgcd(a, m, x, y);
    return d == 1 ? (x + m) % m : -1;
}
```

$a < p$ 且 p 为素数时，有以下两种求法
费马小定理

```
ll inv(ll a, ll p) { return Pow(a, p - 2, p); }
```

贾志鹏线性筛

```
for (int i = 2; i < n; i++) inv[i] = inv[p % i] * (p - p / i) % p;
```

1.3.4 Discrete Logarithm

求解 $a^x \equiv b \pmod p$，p 可以不是质数

```
ll exbsgs(ll a, ll b, ll p)
{
    if (b == 1LL) return 0;
    ll t, d = 1, k = 0;
    while ((t = gcd(a, p)) != 1)
    {
        if (b % t) return -1;
        ++k, b /= t, p /= t, d = d * (a / t) % p;
        if (b == d) return k;
    }
    map<ll, ll> dic;
    ll m = ceil(sqrt(p));
    ll a_m = Pow(a, m, p), mul = b;
    for (ll j = 1; j <= m; ++j) mul = mul * a % p, dic[mul] = j;
    for (ll i = 1; i <= m; ++i)
    {
```

```
        d = d * a_m % p;
        if (dic[d]) return i * m - dic[d] + k;
    }
    return -1;
}
```

1.4 Modulo Linear Equation

1.4.1 Chinese Remainder Theory

$X \equiv r_i \pmod{m_i}$; 要求 m_i 两两互质
引用返回通解 $X = re + k * mo$

```
void crt(ll r[], ll m[], ll n, ll &re, ll &mo)
{
    mo = 1, re = 0;
    for (int i = 0; i < n; i++) mo *= m[i];
    for (int i = 0; i < n; i++)
    {
        ll x, y,  tm = mo / m[i];
        ll d = exgcd(tm, m[i], x, y);
        re = (re + tm * x * r[i]) % mo;
    }
    re = (re + mo) % mo;
}
```

1.4.2 ExCRT

$X \equiv r_i \pmod{m_i}$; m_i 可以不两两互质
引用返回通解 $X = re + k * mo$; 函数返回是否有解

```
bool excrt(ll r[], ll m[], ll n, ll &re, ll &mo)
{
    ll x, y;
    mo = m[0], re = r[0];
    for (int i = 1; i < n; i++)
    {
        ll d = exgcd(mo, m[i],  x, y);
        if ((r[i] - re) % d != 0) return 0;
        x = (r[i] - re) / d * x % (m[i] / d);
        re += x * mo;
        mo = mo / d * m[i];
        re %= mo;
    }
    re = (re + mo) % mo;
    return 1;
}
```

1.5 Combinatorics

1.5.1 Combination

$0 \le m \le n \le 1000$

```
const int maxn = 1010;
ll C[maxn][maxn];
void CalComb()
```

```
{
    C[0][0] = 1;
    for (int i = 1; i < maxn; i++)
    {
        C[i][0] = 1;
        for (int j = 1; j <= i; j++) C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % mod;
    }
}
```

$0 \le m \le n \le 10^5$, 模 p 为素数

```
const int maxn = 100010;
ll f[maxn];
ll inv[maxn]; // 阶乘的逆元
void CalFact()
{
    f[0] = 1;
    for (int i = 1; i < maxn; i++) f[i] = (f[i - 1] * i) % p;
    inv[maxn - 1] = Pow(f[maxn - 1], p - 2, p);
    for (int i = maxn - 2; ~i; i--) inv[i] = inv[i + 1] * (i + 1) % p;
}
ll C(int n, int m) { return f[n] * inv[m] % p * inv[n - m] % p; }
```

1.5.2 Lucas

$1 \le n, m \le 1000000000, 1 < p < 100000$, p 是素数

```
const int maxp = 100010;
ll f[maxn];
ll inv[maxn]; // 阶乘的逆元
void CalFact()
{
    f[0] = 1;
    for (int i = 1; i < maxn; i++) f[i] = (f[i - 1] * i) % p;
    inv[maxn - 1] = Pow(f[maxn - 1], p - 2, p);
    for (int i = maxn - 2; ~i; i--) inv[i] = inv[i + 1] * (i + 1) % p;
}
ll Lucas(ll n, ll m, ll p)
{
    ll ret = 1;
    while (n && m)
    {
        ll a = n % p, b = m % p;
        if (a < b) return 0;
        ret = ret * f[a] % p * inv[b] % p * inv[a - b] % p;
        n /= p, m /= p;
    }
    return ret;
}
```

1.5.3 Big Combination

$0 \le n \le 10^9, 0 \le m \le 10^4, 1 \le k \le 10^9 + 7$

```
vector<int> v;
int dp[110];
ll Cal(int l, int r, int k, int dis)
{
```

```cpp
    ll res = 1;
    for (int i = l; i <= r; i++)
    {
        int t = i;
        for (int j = 0; j < v.size(); j++)
        {
            int y = v[j];
            while (t % y == 0) dp[j] += dis, t /= y;
        }
        res = res * (ll)t % k;
    }
    return res;
}
ll Comb(int n, int m, int k)
{
    memset(dp, 0, sizeof(dp));
    v.clear();
    int tmp = k;
    for (int i = 2; i * i <= tmp; i++)
        if (tmp % i == 0)
        {
            int num = 0;
            while (tmp % i == 0) tmp /= i, num++;
            v.push_back(i);
        }
    if (tmp != 1) v.push_back(tmp);
    ll ans = Cal(n - m + 1, n, k, 1);
    for (int j = 0; j < v.size(); j++) ans = ans * Pow(v[j], dp[j], k) % k;
    ans = ans * inv(Cal(2, m, k, -1), k) % k;
    return ans;
}
```

1.5.4 Polya

推论：一共 n 个置换，第 i 个置换的循环节个数为 $gcd(i,n)$

$N*N$ 的正方形格子，$c^{n^2}+2c^{\frac{n^2+3}{4}}+c^{\frac{n^2+1}{2}}+2c^{n\frac{n+1}{2}}+2c^{\frac{n(n+1)}{2}}$

正六面体，$\frac{m^8+17m^4+6m^2}{24}$ 正四面体，$\frac{m^4+11m^2}{12}$

长度为 n 的项链串用 c 种颜色染 $\sum_{d|n}\frac{\varphi(n/d)c^d}{n}$

```cpp
ll solve(int c, int n)
{
    if (n == 0) return 0;
    ll ans = 0;
    for (int i = 1; i <= n; i++) ans += Pow(c, __gcd(i, n));
    if (n & 1) ans += n * Pow(c, n + 1 >> 1);
    else ans += n / 2 * (1 + c) * Pow(c, n >> 1);
    return ans / n / 2;
}
```

每种颜色至少涂多少个，求方案数

```cpp
ll polya(int a)//a为循环节长度
{
    ll dp[65][65] = {0}; //前者为颜色，后者为未填充格子个数
    int tot = 60 / a, limit = 0;
    dp[0][tot] = 1;
    for (int i = 1; i <= n; i++)
```

```
    {
        int tmp = (c[i] + a - 1) / a;
        int up2 = tot - limit;
        int up1 = up2 - tmp;           //最多空tot-(limit + tmp)
        for (int j = 0; j <= up1; j++) //最少空0个，即填满
        {
            for (int k = tmp; j + k <= up2; k++) //至少选tmp个，最多选tot - limit -j
                (dp[i][j] += dp[i - 1][j + k] * C[j + k][k]) %= p;
        }
        limit += tmp;
    }
    return dp[n][0];
}
```

每种颜色要有多少个，求恰好满足的方案数

```
bool check(int b) //a[i]是每种颜色有多少个，b是循环节长度
{
    for (int i = 0; i < n; i++)
        if (a[i] % b) return false;
    return true;
}
ll solve(int tot, int b) //tot是总数，b是循环节长度
{
    if (!check(b)) return 0;
    ll res = 1, cnt = tot / b; //cnt循环节个数
    for (int i = 0; i < 6; i++)
    {
        res *= C[cnt][a[i] / b];
        cnt -= a[i] / b;
    }
    return res;
}
```

1.6 Fast Power

```
inline ll Mul(ll a, ll b, ll m)
{
    if (m <= 1000000000)
        return a * b % m;
    else if (m <= 1000000000000ll)
        return (((a * (b >> 20) % m) << 20) + (a * (b & ((1 << 20) - 1)))) % m;
    else
    {
        ll d = (ll)floor(a * (long double)b / m + 0.5);
        ll ret = (a * b - d * m) % m;
        if (ret < 0) ret += m;
        return ret;
    }
}
ll Pow(ll a, ll n, ll m)
{
    ll t = 1;
    for (; n; n >>= 1, a = (a * a % m))
        if (n & 1) t = (t * a % m);
    return t;
}
```

1.7 Mobius Inversion

1.7.1 Mobius

$F(n) = \sum_{d|n} f(d) \Rightarrow f(n) = \sum_{d|n} \mu(d)F(\frac{n}{d})$
$F(n) = \sum_{n|d} f(d) \Rightarrow f(n) = \sum_{n|d} \mu(\frac{d}{n})F(d)$

```
const int maxn = "Edit";
int prime[maxn], tot, mu[maxn];
bool check[maxn];
void CalMu()
{
    mu[1] = 1;
    for (int i = 2; i < maxn; i++)
    {
        if (!check[i]) prime[tot++] = i, mu[i] = -1;
        for (int j = 0; j < tot; j++)
        {
            if (i * prime[j] >= maxn) break;
            check[i * prime[j]] = true;
            if (i % prime[j] == 0)
            {
                mu[i * prime[j]] = 0;
                break;
            }
            else
                mu[i * prime[j]] = -mu[i];
        }
    }
}
```

1.7.2 Examples

有 n 个数 ($n \leq 100000, 1 \leq a_i \leq 10^6$)，问这 n 个数中互质的数的对数

```
const int maxn = "Edit";
int b[maxn];
ll solve(int n)
{
    ll ans = 0;
    for (int i = 0, x; i < n; i++) scanf("%d", &x), b[x]++;
    for (int i = 1; i < maxn; i++)
    {
        int cnt = 0;
        for (int j = i; j < maxn; j += i) cnt += b[j];
        ans += 1LL * mu[i] * cnt * cnt;
    }
    return (ans - b[1]) / 2;
}
```

$\gcd(x, y) = 1$ 的对数, $x \leq n, y \leq m$

```
ll solve(int n, int m)
{
    if (n > m) swap(n, m);
    ll ans = 0;
    for (int i = 1; i <= n; i++) ans += (ll)mu[i] * (n / i) * (m / i);
    /*
    数论分块写法(sum为莫比乌斯函数的前缀和)
```

```
    for (int i = 1; i <= n; i = pos + 1)
    {
        pos = min(n / (n / i), m / (m / i));
        ans += 1LL * (sum[pos] - sum[i - 1]) * (n / i) * (m / i);
    }
    */
    return ans;
}
```

1.8 Fast Transformation

1.8.1 FFT

```
const double PI = acos(-1.0);
//复数结构体
struct Complex
{
    double x, y; //实部和虚部 x+yi
    Complex(double _x = 0.0, double _y = 0.0) { x = _x, y = _y; }
    Complex operator-(const Complex& b) const { return Complex(x - b.x, y - b.y); }
    Complex operator+(const Complex& b) const { return Complex(x + b.x, y + b.y); }
    Complex operator*(const Complex& b) const { return Complex(x * b.x - y * b.y, x * b.y + y * b.x); }
};
void change(Complex y[], int len)
{
    for (int i = 1, j = len / 2; i < len - 1; i++)
    {
        if (i < j) swap(y[i], y[j]);
        int k = len / 2;
        while (j >= k) j -= k, k /= 2;
        if (j < k) j += k;
    }
}
/*
* len必须为2^k形式，
* on==1时是DFT，on==-1时是IDFT
*/
void fft(Complex y[], int len, int on)
{
    change(y, len);
    for (int h = 2; h <= len; h <<= 1)
    {
        Complex wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));
        for (int j = 0; j < len; j += h)
        {
            Complex w(1, 0);
            for (int k = j; k < j + h / 2; k++)
            {
                Complex u = y[k];
                Complex t = w * y[k + h / 2];
                y[k] = u + t, y[k + h / 2] = u - t;
                w = w * wn;
            }
        }
    }
    if (on == -1)
        for (int i = 0; i < len; i++) y[i].x /= len;
}
```

1.8.2 NTT

模数 P 为费马素数，G 为 P 的原根。$G^{\frac{P-1}{n}}$ 具有和 $w_n = e^{\frac{2i\pi}{n}}$ 相似的性质。具体的 P 和 G 可参考 1.11

```
const int mod = 119 << 23 | 1;
const int G = 3;
int wn[20];
void getwn()
{ //  千万不要忘记
    for (int i = 0; i < 20; i++) wn[i] = Pow(G, (mod - 1) / (1 << i), mod);
}
void change(int y[], int len)
{
    for (int i = 1, j = len / 2; i < len - 1; i++)
    {
        if (i < j) swap(y[i], y[j]);
        int k = len / 2;
        while (j >= k) j -= k, k /= 2;
        if (j < k) j += k;
    }
}
void ntt(int y[], int len, int on)
{
    change(y, len);
    for (int h = 2, id = 1; h <= len; h <<= 1, id++)
    {
        for (int j = 0; j < len; j += h)
        {
            int w = 1;
            for (int k = j; k < j + h / 2; k++)
            {
                int u = y[k] % mod;
                int t = 1LL * w * (y[k + h / 2] % mod) % mod;
                y[k] = (u + t) % mod, y[k + h / 2] = ((u - t) % mod + mod) % mod;
                w = 1LL * w * wn[id] % mod;
            }
        }
    }
    if (on == -1)
    {
        //  原本的除法要用逆元
        int inv = Pow(len, mod - 2, mod);
        for (int i = 1; i < len / 2; i++) swap(y[i], y[len - i]);
        for (int i = 0; i < len; i++) y[i] = 1LL * y[i] * inv % mod;
    }
}
```

1.8.3 FWT

```
void fwt(int f[], int m)
{
    int n = __builtin_ctz(m);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            if (j & (1 << i))
            {
                int l = f[j ^ (1 << i)], r = f[j];
                f[j ^ (1 << i)] = l + r, f[j] = l - r;
```

```
                    // or: f[j] += f[j ^ (1 << i)];
                    // and: f[j ^ (1 << i)] += f[j];
                }
}
void ifwt(int f[], int m)
{
    int n = __builtin_ctz(m);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            if (j & (1 << i))
            {
                int l = f[j ^ (1 << i)], r = f[j];
                f[j ^ (1 << i)] = (l + r) / 2, f[j] = (l - r) / 2;
                // 如果有取模需要使用逆元
                // or: f[j] -= f[j ^ (1 << i)];
                // and: f[j ^ (1 << i)] -= f[j];
            }
}
```

1.9 Numerical Integration

1.9.1 Adaptive Simpson's Rule

$\int_a^b f(x)dx \approx \frac{b-a}{6}[f(a) + 4f(\frac{a+b}{2}) + f(b)]$
$|S(a,c) + S(c,b) - S(a,b)|/15 < \epsilon$

```
double F(double x) {}
double simpson(double a, double b)
{ // 三点Simpson法
    double c = a + (b - a) / 2;
    return (F(a) + 4 * F(c) + F(b)) * (b - a) / 6;
}
double asr(double a, double b, double eps, double A)
{ //自适应Simpson公式（递归过程）。已知整个区间[a,b]上的三点Simpson值A
    double c = a + (b - a) / 2;
    double L = simpson(a, c), R = simpson(c, b);
    if (fabs(L + R - A) <= 15 * eps) return L + R + (L + R - A) / 15.0;
    return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
}
double asr(double a, double b, double eps) { return asr(a, b, eps, simpson(a, b)); }
```

1.9.2 Berlekamp-Massey

```
const int maxn = 1 << 14;
ll res[maxn], base[maxn], _c[maxn], _md[maxn];
vector<int> Md;
void mul(ll* a, ll* b, int k)
{
    for (int i = 0; i < k + k; i++) _c[i] = 0;
    for (int i = 0; i < k; i++)
        if (a[i])
            for (int j = 0; j < k; j++) _c[i + j] = (_c[i + j] + a[i] * b[j]) % mod;
    for (int i = k + k - 1; i >= k; i--)
        if (_c[i])
            for (int j = 0; j < Md.size(); j++) _c[i - k + Md[j]] = (_c[i - k + Md[j]]
    - _c[i] * _md[Md[j]]) % mod;
    for (int i = 0; i < k; i++) a[i] = _c[i];
}
```

```
int solve(ll n, VI a, VI b)
{
    ll ans = 0, pnt = 0;
    int k = a.size();
    assert(a.size() == b.size());
    for (int i = 0; i < k; i++) _md[k - 1 - i] = -a[i];
    _md[k] = 1;
    Md.clear();
    for (int i = 0; i < k; i++)
        if (_md[i] != 0) Md.push_back(i);
    for (int i = 0; i < k; i++) res[i] = base[i] = 0;
    res[0] = 1;
    while ((1LL << pnt) <= n) pnt++;
    for (int p = pnt; p >= 0; p--)
    {
        mul(res, res, k);
        if ((n >> p) & 1)
        {
            for (int i = k - 1; i >= 0; i--) res[i + 1] = res[i];
            res[0] = 0;
            for (int j = 0; j < Md.size(); j++) res[Md[j]] = (res[Md[j]] - res[k] * _md
    [Md[j]]) % mod;
        }
    }
    for (int i = 0; i < k; i++) ans = (ans + res[i] * b[i]) % mod;
    if (ans < 0) ans += mod;
    return ans;
}
VI BM(VI s)
{
    VI C(1, 1), B(1, 1);
    int L = 0, m = 1, b = 1;
    for (int n = 0; n < s.size(); n++)
    {
        ll d = 0;
        for (int i = 0; i <= L; i++) d = (d + (ll)C[i] * s[n - i]) % mod;
        if (d == 0)
            ++m;
        else if (2 * L <= n)
        {
            VI T = C;
            ll c = mod - d * Pow(b, mod - 2) % mod;
            while (C.size() < B.size() + m) C.push_back(0);
            for (int i = 0; i < B.size(); i++) C[i + m] = (C[i + m] + c * B[i]) % mod;
            L = n + 1 - L, B = T, b = d, m = 1;
        }
        else
        {
            ll c = mod - d * Pow(b, mod - 2) % mod;
            while (C.size() < B.size() + m) C.push_back(0);
            for (int i = 0; i < B.size(); i++) C[i + m] = (C[i + m] + c * B[i]) % mod;
            ++m;
        }
    }
    return C;
}
int gao(VI a, ll n)
{
    VI c = BM(a);
```

```cpp
    c.erase(c.begin());
    for (int i = 0; i < c.size(); i++) c[i] = (mod - c[i]) % mod;
    return solve(n, c, VI(a.begin(), a.begin() + c.size()));
}
```

1.9.3 Simplex

输入矩阵 a 描述线性规划的标准形式。

a 为 $m+1$ 行 $n+1$ 列，其中行 $0 \sim m-1$ 为不等式，行 m 为目标函数（最大化）。

列 $0 \sim n-1$ 为变量 $0 \sim n-1$ 的系数，列 n 为常数项。

约束为 $a_{i,0}x_0 + a_{i,1}x_1 + \cdots \leq a_{i,n}$，目标为 $\max(a_{m,0}x_0 + a_{m,1}x_1 + \cdots + a_{m,n-1}x_{n-1} - a_{m,n})$

注意：变量均有非负约束 $x[i] \geq 0$

```cpp
const int maxm = 500; // 约束数目上限
const int maxn = 500; // 变量数目上限
const double INF = 1e100;
const double eps = 1e-10;
struct Simplex
{
    int n;                  // 变量个数
    int m;                  // 约束个数
    double a[maxm][maxn]; // 输入矩阵
    int B[maxm], N[maxn]; // 算法辅助变量
    void pivot(int r, int c)
    {
        swap(N[c], B[r]);
        a[r][c] = 1 / a[r][c];
        for (int j = 0; j <= n; j++)
            if (j != c) a[r][j] *= a[r][c];
        for (int i = 0; i <= m; i++)
            if (i != r)
            {
                for (int j = 0; j <= n; j++)
                    if (j != c) a[i][j] -= a[i][c] * a[r][j];
                a[i][c] = -a[i][c] * a[r][c];
            }
    }
    bool feasible()
    {
        for (;;)
        {
            int r, c;
            double p = INF;
            for (int i = 0; i < m; i++)
                if (a[i][n] < p) p = a[r = i][n];
            if (p > -eps) return true;
            p = 0;
            for (int i = 0; i < n; i++)
                if (a[r][i] < p) p = a[r][c = i];
            if (p > -eps) return false;
            p = a[r][n] / a[r][c];
            for (int i = r + 1; i < m; i++)
                if (a[i][c] > eps)
                {
```

```
                    double v = a[i][n] / a[i][c];
                    if (v < p) r = i, p = v;
                }
            pivot(r, c);
        }
    }
    // 解有界返回1，无解返回0，无界返回-1。b[i]为x[i]的值，ret为目标函数的值
    int simplex(int n, int m, double x[maxn], double& ret)
    {
        this->n = n, this->m = m;
        for (int i = 0; i < n; i++) N[i] = i;
        for (int i = 0; i < m; i++) B[i] = n + i;
        if (!feasible()) return 0;
        for (;;)
        {
            int r, c;
            double p = 0;
            for (int i = 0; i < n; i++)
                if (a[m][i] > p) p = a[m][c = i];
            if (p < eps)
            {
                for (int i = 0; i < n; i++)
                    if (N[i] < n) x[N[i]] = 0;
                for (int i = 0; i < m; i++)
                    if (B[i] < n) x[B[i]] = a[i][n];
                ret = -a[m][n];
                return 1;
            }
            p = INF;
            for (int i = 0; i < m; i++)
                if (a[i][c] > eps)
                {
                    double v = a[i][n] / a[i][c];
                    if (v < p) r = i, p = v;
                }
            if (p == INF) return -1;
            pivot(r, c);
        }
    }
};
```

1.10 Others

约瑟夫问题

n 个人围成一圈，从第一个开始报数，第 m 个将被杀掉

```
int josephus(int n, int m)
{
    int r = 0;
    for (int k = 1; k <= n; ++k) r = (r + m) % k;
    return r + 1;
}
```

n^n 最左边一位数

```
int leftmost(int n)
{
    double m = n * log10((double)n);
    double g = m - (ll)m;
```

```
    return (int)pow(10.0, g);
}
```

$n!$ 位数

```
int count(ll n)
{
    if (n == 1) return 1;
    return (int)ceil(0.5 * log10(2 * M_PI * n) + n * log10(n) - n * log10(M_E));
}
```

1.11 Formula

1. 约数定理：若 $n = \prod_{i=1}^{k} p_i^{a_i}$，则
   (a) 约数个数 $f(n) = \prod_{i=1}^{k}(a_i + 1)$
   (b) 约数和 $g(n) = \prod_{i=1}^{k}(\sum_{j=0}^{a_i} p_i^j)$
2. 小于 n 且互素的数之和为 $n\varphi(n)/2$
3. 若 $\gcd(n, i) = 1$，则 $\gcd(n, n-i) = 1 (1 \le i \le n)$
4. 错排公式：$D(n) = (n-1)(D(n-2) + D(n-1)) = \sum_{i=2}^{n} \frac{(-1)^k n!}{k!} = [\frac{n!}{e} + 0.5]$
5. 威尔逊定理：$p\ is\ prime\ \Rightarrow (p-1)! \equiv -1 \pmod p$
6. 欧拉定理：$\gcd(a, n) = 1 \Rightarrow a^{\varphi(n)} \equiv 1 \pmod n$
7. 欧拉定理推广：$\gcd(n, p) = 1 \Rightarrow a^n \equiv a^{n\%\varphi(p)} \pmod p$
8. 模的幂公式：$a^n \pmod m = \begin{cases} a^n \mod m & n < \varphi(m) \\ a^{n\%\varphi(m)+\varphi(m)} \mod m & n \ge \varphi(m) \end{cases}$
9. 素数定理：对于不大于 n 的素数个数 $\pi(n)$，$\lim\limits_{n\to\infty} \pi(n) = \frac{n}{\ln n}$
10. 位数公式：正整数 x 的位数 $N = \log_{10}(n) + 1$
11. 斯特灵公式 $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$
12. 设 $a > 1, m, n > 0$, 则 $\gcd(a^m - 1, a^n - 1) = a^{\gcd(m,n)} - 1$
13. 设 $a > b, \gcd(a, b) = 1$, 则 $\gcd(a^m - b^m, a^n - b^n) = a^{\gcd(m,n)} - b^{\gcd(m,n)}$

$$G = \gcd(C_n^1, C_n^2, ..., C_n^{n-1}) = \begin{cases} n, & n \text{ is prime} \\ 1, & n \text{ has multy prime factors} \\ p, & n \text{ has single prime factor } p \end{cases}$$

   $\gcd(Fib(m), Fib(n)) = Fib(\gcd(m, n))$
14. 若 $\gcd(m, n) = 1$, 则:
   (a) 最大不能组合的数为 $m * n - m - n$
   (b) 不能组合数个数 $N = \frac{(m-1)(n-1)}{2}$
15. $(n+1)lcm(C_n^0, C_n^1, ..., C_n^{n-1}, C_n^n) = lcm(1, 2, ..., n+1)$
16. 若 p 为素数，则 $(x + y + ... + w)^p \equiv x^p + y^p + ... + w^p \pmod p$
17. 卡特兰数：1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012
   $h(0) = h(1) = 1, h(n) = \frac{(4n-2)h(n-1)}{n+1} = \frac{C_{2n}^n}{n+1} = C_{2n}^n - C_{2n}^{n-1}$
18. 伯努利数：$B_n = -\frac{1}{n+1}\sum_{i=0}^{n-1} C_{n+1}^i B_i$

$$\sum_{i=1}^{n} i^k = \frac{1}{k+1}\sum_{i=1}^{k+1} C_{k+1}^i B_{k+1-i}(n+1)^i$$

19. 二项式反演：

$$f_n = \sum_{i=0}^{n} (-1)^i \binom{n}{i} g_i \Leftrightarrow g_n = \sum_{i=0}^{n} (-1)^i \binom{n}{i} f_i$$

$$f_n = \sum_{i=0}^{n} \binom{n}{i} g_i \Leftrightarrow g_n = \sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} f_i$$

20. FFT 常用素数

$r\ 2^k + 1$	r	k	g
3	1	1	2
5	1	2	2
17	1	4	3
97	3	5	5
193	3	6	5
257	1	8	3
7681	15	9	17
12289	3	12	11
40961	5	13	3
65537	1	16	3
786433	3	18	10
5767169	11	19	3
7340033	7	20	3
23068673	11	21	3
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
998244353	119	23	3
1004535809	479	21	3
2013265921	15	27	31
2281701377	17	27	3
3221225473	3	30	5
75161927681	35	31	3
77309411329	9	33	7
206158430209	3	36	22
2061584302081	15	37	7
2748779069441	5	39	3
6597069766657	3	41	5
39582418599937	9	42	5
79164837199873	9	43	5
263882790666241	15	44	7
1231453023109121	35	45	3
1337006139375617	19	46	3
3799912185593857	27	47	5
4222124650659841	15	48	19
7881299347898369	7	50	6
31525197391593473	7	52	3
180143985094819841	5	55	6
1945555039024054273	27	56	5
4179340454199820289	29	57	3

2 String Processing

2.1 KMP

```
// 返回y中x的个数
const int N = "Edit";
int next[N];
void initkmp(char x[], int m)
{
    int i = 0, j = next[0] = -1;
    while (i < m)
    {
        while (j != -1 && x[i] != x[j]) j = next[j];
        next[++i] = ++j;
    }
}
int kmp(char x[], int m, char y[], int n)
{
    int i, j, ans;
    i = j = ans = 0;
    initkmp(x, m);
    while (i < n)
    {
        while (j != -1 && y[i] != x[j]) j = next[j];
        i++, j++;
        if (j >= m) ans++, j = next[j];
    }
    return ans;
}
```

2.2 ExtendKMP

```
//next[i]:x[i...m-1]与x[0...m-1]的最长公共前缀
//extend[i]:y[i...n-1]与x[0...m-1]的最长公共前缀
const int N = "Edit";
int next[N], extend[N];
void pre_ekmp(char x[], int m)
{
    next[0] = m;
    int j = 0;
    while (j + 1 < m && x[j] == x[j + 1]) j++;
    next[1] = j;
    int k = 1;
    for (int i = 2; i < m; i++)
    {
        int p = next[k] + k - 1;
        int L = next[i - k];
        if (i + L < p + 1)
            next[i] = L;
        else
        {
            j = max(0, p - i + 1);
            while (i + j < m && x[i + j] == x[j]) j++;
            next[i] = j;
            k = i;
        }
    }
}
```

```
void ekmp(char x[], int m, char y[], int n)
{
    pre_ekmp(x, m, next);
    int j = 0;
    while (j < n && j < m && x[j] == y[j]) j++;
    extend[0] = j;
    int k = 0;
    for (int i = 1; i < n; i++)
    {
        int p = extend[k] + k - 1;
        int L = next[i - k];
        if (i + L < p + 1)
            extend[i] = L;
        else
        {
            j = max(0, p - i + 1);
            while (i + j < n && j < m && y[i + j] == x[j]) j++;
            extend[i] = j, k = i;
        }
    }
}
```

2.3 Manacher

$O(n)$ 求解最长回文子串

```
const int N = "Edit";
char s[N], str[N << 1];
int p[N << 1];
void Manacher(char s[], int& n)
{
    str[0] = '$', str[1] = '#';
    for (int i = 0; i < n; i++) str[(i << 1) + 2] = s[i], str[(i << 1) + 3] = '#';
    n = 2 * n + 2;
    str[n] = 0;
    int mx = 0, id;
    for (int i = 1; i < n; i++)
    {
        p[i] = mx > i ? min(p[2 * id - i], mx - i) : 1;
        while (str[i - p[i]] == str[i + p[i]]) p[i]++;
        if (p[i] + i > mx) mx = p[i] + i, id = i;
    }
}
int solve(char s[])
{
    int n = strlen(s);
    Manacher(s, n);
    return *max_elememt(p, p + n) - 1;
}
```

2.4 Aho-Corasick Automaton

```
const int maxn = "Edit";
struct Trie
{
    int ch[maxn][26], f[maxn], val[maxn];
    int sz, rt;
```

```
    int newnode() { memset(ch[sz], -1, sizeof(ch[sz])), val[sz] = 0; return sz++; }
    void init() { sz = 0, rt = newnode(); }
    inline int idx(char c) { return c - 'A'; };
    void insert(const char* s)
    {
        int u = 0;
        for (int i = 0; s[i]; i++)
        {
            int c = idx(s[i]);
            if (ch[u][c] == -1) ch[u][c] = newnode();
            u = ch[u][c];
        }
        val[u]++;
    }
    void build()
    {
        queue<int> q;
        f[rt] = rt;
        for (int c = 0; c < 26; c++)
        {
            if (~ch[rt][c])
                f[ch[rt][c]] = rt, q.push(ch[rt][c]);
            else
                ch[rt][c] = rt;
        }
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            // val[u] |= val[f[u]];
            for (int c = 0; c < 26; c++)
            {
                if (~ch[u][c])
                    f[ch[u][c]] = ch[f[u]][c], q.push(ch[u][c]);
                else
                    ch[u][c] = ch[f[u]][c];
            }
        }
    }
    //返回主串中有多少模式串
    int query(const char* s)
    {
        int u = rt;
        int res = 0;
        for (int i = 0; s[i]; i++)
        {
            int c = idx(s[i]);
            u = ch[u][c];
            int tmp = u;
            while (tmp != rt)
            {
                res += val[tmp];
                val[tmp] = 0;
                tmp = f[tmp];
            }
        }
        return res;
    }
};
```

2.5 Suffix Array

```
//倍增算法构造后缀数组,复杂度O(nlogn)
const int maxn = "Edit";
struct Suffix_Array
{
    char s[maxn];
    int sa[maxn], t[maxn], t2[maxn], c[maxn], rank[maxn], height[maxn];
    void build_sa(int m, int n)
    { //n为字符串的长度,字符集的值为0~m-1
        n++;
        int *x = t, *y = t2;
        //基数排序
        for (int i = 0; i < m; i++) c[i] = 0;
        for (int i = 0; i < n; i++) c[x[i] = s[i]]++;
        for (int i = 1; i < m; i++) c[i] += c[i - 1];
        for (int i = n - 1; ~i; i--) sa[--c[x[i]]] = i;
        for (int k = 1; k <= n; k <<= 1)
        { //直接利用sa数组排序第二关键字
            int p = 0;
            for (int i = n - k; i < n; i++) y[p++] = i;
            for (int i = 0; i < n; i++)
                if (sa[i] >= k) y[p++] = sa[i] - k;
            //基数排序第一关键字
            for (int i = 0; i < m; i++) c[i] = 0;
            for (int i = 0; i < n; i++) c[x[y[i]]]++;
            for (int i = 0; i < m; i++) c[i] += c[i - 1];
            for (int i = n - 1; ~i; i--) sa[--c[x[y[i]]]] = y[i];
            //根据sa和y数组计算新的x数组
            swap(x, y);
            p = 1;
            x[sa[0]] = 0;
            for (int i = 1; i < n; i++)
                x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k]
 ? p - 1 : p++;
            if (p >= n) break; //以后即使继续倍增, sa也不会改变, 推出
            m = p;             //下次基数排序的最大值
        }
        n--;
        int k = 0;
        for (int i = 0; i <= n; i++) rank[sa[i]] = i;
        for (int i = 0; i < n; i++)
        {
            if (k) k--;
            int j = sa[rank[i] - 1];
            while (s[i + k] == s[j + k]) k++;
            height[rank[i]] = k;
        }
    }

    int dp[maxn][30];
    void initrmq(int n)
    {
        for (int i = 1; i <= n; i++)
            dp[i][0] = height[i];
        for (int j = 1; (1 << j) <= n; j++)
            for (int i = 1; i + (1 << j) - 1 <= n; i++)
                dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
    }
```

```
    int rmq(int l, int r)
    {
        int k = 31 - __builtin_clz(r - l + 1);
        return min(dp[l][k], dp[r - (1 << k) + 1][k]);
    }
    int lcp(int a, int b)
    { // 求两个后缀的最长公共前缀
        a = rank[a], b = rank[b];
        if (a > b) swap(a, b);
        return rmq(a + 1, b);
    }
};
```

2.6 Suffix Automation

```
const int maxn = "Edit";
struct SAM
{
    int len[maxn << 1], link[maxn << 1], ch[maxn << 1][26];
    int num[maxn << 1]; //每个结点所代表的字符串的出现次数
    int sz, rt, last;
    int newnode(int x = 0)
    {
        len[sz] = x;
        link[sz] = -1;
        memset(ch[sz], -1, sizeof(ch[sz]));
        return sz++;
    }
    void init() { sz = last = 0, rt = newnode(); }
    void reset() { last = 0; }
    void extend(int c)
    {
        int np = newnode(len[last] + 1);
        int p;
        for (p = last; ~p && ch[p][c] == -1; p = link[p]) ch[p][c] = np;
        if (p == -1)
            link[np] = rt;
        else
        {
            int q = ch[p][c];
            if (len[p] + 1 == len[q])
                link[np] = q;
            else
            {
                int nq = newnode(len[p] + 1);
                memcpy(ch[nq], ch[q], sizeof(ch[q]));
                link[nq] = link[q], link[q] = link[np] = nq;
                for (; ~p && ch[p][c] == q; p = link[p]) ch[p][c] = nq;
            }
        }
        last = np;
    }
    int topcnt[maxn], topsam[maxn << 1];
    void build(const char* s)
    { // 加入串后拓扑排序
        memset(topcnt, 0, sizeof(topcnt));
        for (int i = 0; i < sz; i++) topcnt[len[i]]++;
        for (int i = 0; i < maxn - 1; i++) topcnt[i + 1] += topcnt[i];
```

```
        for (int i = 0; i < sz; i++) topsam[--topcnt[len[i]]] = i;
        int u = rt;
        for (int i = 0; s[i]; i++) num[u = ch[u][s[i] - 'a']] = 1;
        for (int i = sz - 1; ~i; i--)
        {
            int u = topsam[i];
            if (~link[u]) num[link[u]] += num[u];
        }
    }
};
```

2.7 Palindromic Tree

```
const int maxn = "Edit";
struct Palindromic_Tree
{
    int ch[maxn][26], f[maxn], len[maxn], s[maxn];
    int cnt[maxn]; // 结点表示的本质不同的回文串的个数(调用count()后)
    int num[maxn]; // 结点表示的最长回文串的最右端点为回文串结尾的回文串个数
    int last, sz, n;
    int newnode(int x)
    {
        memset(ch[sz], 0, sizeof(ch[sz]));
        cnt[sz] = num[sz] = 0, len[sz] = x;
        return sz++;
    }
    void init()
    {
        sz = 0;
        newnode(0), newnode(-1);
        last = n = 0, s[0] = -1, f[0] = 1;
    }
    int get_fail(int u)
    {
        while (s[n - len[u] - 1] != s[n]) u = f[u];
        return u;
    }
    void add(int c)
    { // c-='a'
        s[++n] = c;
        int u = get_fail(last);
        if (!ch[u][c])
        {
            int np = newnode(len[u] + 2);
            f[np] = ch[get_fail(f[u])][c];
            num[np] = num[f[np]] + 1;
            ch[u][c] = np;
        }
        last = ch[u][c];
        cnt[last]++;
    }
    void count()
    {
        for (int i = sz - 1; ~i; i--) cnt[f[i]] += cnt[i];
    }
};
```

2.8 Hash

```
typedef unsigned long long ull;
const ull Seed_Pool[] = {146527, 19260817};
const ull Mod_Pool[] = {1000000009, 998244353};
struct Hash
{
    ull SEED, MOD;
    vector<ull> p, h;
    Hash() {}
    Hash(const string& s, const int& seed_index, const int& mod_index)
    {
        SEED = Seed_Pool[seed_index];
        MOD = Mod_Pool[mod_index];
        int n = s.length();
        p.resize(n + 1), h.resize(n + 1);
        p[0] = 1;
        for (int i = 1; i <= n; i++) p[i] = p[i - 1] * SEED % MOD;
        for (int i = 1; i <= n; i++) h[i] = (h[i - 1] * SEED % MOD + s[i - 1]) % MOD;
    }
    ull get(int l, int r) { return (h[r] - h[l] * p[r - l] % MOD + MOD) % MOD; }
    ull substr(int l, int m) { return get(l, l + m); }
};
```

3 Data Structure

3.1 Binary Indexed Tree

$O(\log n)$ 查询和修改数组的前缀和

```
// 注意下标应从1开始
template <class T>
struct BIT
{
    vector<T> bit;
    int n;
    void init(int n)
    {
        this->n = n;
        bit.assign(n + 1, 0);
    }
    void update(int x, T v)
    {
        for (; x <= n; x + = x & -x) bit[x] += v
    }
    void query(int x)
    {
        T ret = 0;
        for (; x; x -= x & -x) ret += bit[x];
        return ret;
    }
    // 做权值树状数组时求第k小
    int kth(int k)
    {
        int ret = 0, cnt = 0;
        for (int i = 20; ~i; i--)
        {
            ret ^= (1 << i);
            if (ret > n || cnt + bit[ret] >= k)
                ret ^= (1 << i);
            else
                cnt += bit[ret];
        }
        return ret + 1;
    }
};
```

3.2 Segment Tree

线段树必须要能够裸写，此处仅留矩形面积周长系列备忘。

3.2.1 Area Combination

```
// 矩形面积并
map<double, int> Hash;
map<int, double> rHash;
struct line
{
    double l, r, h;
    int val;
    line(double l = 0, double r = 0, double h = 0, int val = 0) : l(l), r(r), h(h), val
    (val) {}
```

```
    bool operator<(const line& A) const { return h < A.h; }
};
struct Node
{
    int cover;
    double len;
};
const int maxn = 1000;
Node seg[maxn << 2];
void build(int rt, int l, int r)
{
    seg[rt].cover = seg[rt].len = 0;
    if (l == r) return;
    int mid = l + r >> 1;
    build(lson, l, mid);
    build(rson, mid + 1, r);
}
void pushup(int rt, int l, int r)
{
    if (seg[rt].cover > 0)
        seg[rt].len = rHash[r + 1] - rHash[l]; // [l,r)
    else if (l == r)
        seg[rt].len = 0;
    else
        seg[rt].len = seg[lson].len + seg[rson].len;
}
void update(int rt, int l, int r, int L, int R, int val)
{
    if (L <= l && R >= r)
    {
        seg[rt].cover += val;
        pushup(rt, l, r);
        return;
    }
    int mid = l + r >> 1;
    if (mid >= L) update(lson, l, mid, L, R, val);
    if (mid + 1 <= R) update(rson, mid + 1, r, L, R, val);
    pushup(rt, l, r);
}
int main()
{
    int n, kase = 0;
    while (~scanf("%d", &n))
    {
        if (!n) break;
        double x1, x2, y1, y2;
        vector<line> a;
        set<double> xval;
        for (int i = 0; i < n; i++)
        {
            scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
            a.emplace_back(x1, x2, y1, 1);
            a.emplace_back(x1, x2, y2, -1);
            xval.insert(x1);
            xval.insert(x2);
        }
        // 离散化
        Hash.clear(), rHash.clear();
        int cnt = 0;
```

```
        for (auto& v : xval)
        {
            Hash[v] = ++cnt;
            rHash[cnt] = v;
        }
        sort(a.begin(), a.end());
        build(1, 1, cnt);
        double ans = 0;
        for (int i = 0; i < a.size() - 1; i++)
        {
            update(1, 1, cnt, Hash[a[i].l], Hash[a[i].r] - 1,
                   a[i].val); //[l,r)
            ans += (a[i + 1].h - a[i].h) * seg[1].len;
        }
        printf("Test case #%d\n", ++kase);
        printf("Total explored area: %.2lf\n\n", ans);
    }
}
```

3.2.2 Area Intersection

```
// 矩形面积交
map<double, int> Hash;
map<int, double> rHash;
struct Lines
{
    double l, r, h;
    int val;
    bool operator<(const Lines& A) const { return h < A.h; }
};
struct Node
{
    int cnt;     //  覆盖次数
    double len1; // 覆盖次数大于0的长度
    double len2; // 覆盖次数大于1的长度
};
Node seg[maxn << 2];
void build(int rt, int l, int r)
{
    seg[rt].cnt = seg[rt].len1 = seg[rt].len2 = 0;
    if (l == r) return;
    int mid = l + r >> 1;
    build(lson, l, mid);
    build(rson, mid + 1, r);
}
inline void pushup(int rt, int l, int r)
{
    if (seg[rt].cnt > 1)
        seg[rt].len1 = seg[rt].len2 = rHash[r + 1] - rHash[l];
    else if (seg[rt].cnt == 1)
    {
        seg[rt].len1 = rHash[r + 1] - rHash[l];
        if (l == r)
            seg[rt].len2 = 0;
        else
            seg[rt].len2 = seg[lson].len1 + seg[rson].len1;
    }
    else
```

```
    {
        if (l == r)
            seg[rt].len1 = seg[rt].len2 = 0;
        else
        {
            seg[rt].len1 = seg[lson].len1 + seg[rson].len1;
            seg[rt].len2 = seg[lson].len2 + seg[rson].len2;
        }
    }
}
void update(int rt, int l, int r, int L, int R, int val)
{
    if (L <= l && R >= r)
    {
        seg[rt].cnt += val;
        pushup(rt, l, r);
        return;
    }
    int mid = l + r >> 1;
    if (L <= mid) update(lson, l, mid, L, R, val);
    if (R >= mid + 1) update(rson, mid + 1, r, L, R, val);
    pushup(rt, l, r);
}
int main()
{
    int T;
    scanf("%d", &T);
    while (T--)
    {
        int n;
        scanf("%d", &n);
        double x1, x2, y1, y2;
        vector<Lines> line;
        set<double> X;
        for (int i = 1; i <= n; i++)
        {
            scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
            line.push_back({x1, x2, y1, 1});
            line.push_back({x1, x2, y2, -1});
            X.insert(x1);
            X.insert(x2);
        }
        sort(line.begin(), line.end());
        int cnt = 0;
        Hash.clear();
        rHash.clear();
        for (auto& v : X) Hash[v] = ++cnt, rHash[cnt] = v;
        build(1, 1, cnt);
        double area = 0;
        for (int i = 0; i < line.size() - 1; i++)
        {
            update(1, 1, cnt, Hash[line[i].l], Hash[line[i].r] - 1, line[i].val);
            area += seg[1].len2 * (line[i + 1].h - line[i].h);
        }
        printf("%.2lf\n", area);
    }
}
```

3.2.3 Perimeter Combination

```
// 矩形周长并
int n, m[2];
int sum[maxn << 2], cnt[maxn << 2], all[2][maxn];
struct Seg
{
    int l, r, h, d;
    Seg() {}
    Seg(int l, int r, int h, int d) : l(l), r(r), h(h), d(d) {}
    bool operator<(const Seg& rhs) const { return h < rhs.h; }
} a[2][maxn];
#define lson l, m, rt << 1
#define rson m + 1, r, rt << 1 | 1
void pushup(int p, int l, int r, int rt)
{
    if (cnt[rt])
        sum[rt] = all[p][r + 1] - all[p][l];
    else if (l == r)
        sum[rt] = 0;
    else
        sum[rt] = sum[rt << 1] + sum[rt << 1 | 1];
}
void update(int p, int L, int R, int v, int l, int r, int rt)
{
    if (L <= l && r <= R)
    {
        cnt[rt] += v;
        pushup(p, l, r, rt);
        return;
    }
    int m = l + r >> 1;
    if (L <= m) update(p, L, R, v, lson);
    if (R > m) update(p, L, R, v, rson);
    pushup(p, l, r, rt);
}
int main()
{
    while (scanf("%d", &n) == 1)
    {
        for (int i = 1; i <= n; ++i)
        {
            int x1, y1, x2, y2;
            scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
            all[0][i] = x1, all[0][i + n] = x2;
            all[1][i] = y1, all[1][i + n] = y2;
            a[0][i] = Seg(x1, x2, y1, 1);
            a[0][i + n] = Seg(x1, x2, y2, -1);
            a[1][i] = Seg(y1, y2, x1, 1);
            a[1][i + n] = Seg(y1, y2, x2, -1);
        }
        n <<= 1;
        sort(all[0] + 1, all[0] + 1 + n);
        m[0] = unique(all[0] + 1, all[0] + 1 + n) - all[0] - 1;
        sort(all[1] + 1, all[1] + 1 + n);
        m[1] = unique(all[1] + 1, all[1] + 1 + n) - all[1] - 1;
        sort(a[0] + 1, a[0] + 1 + n);
        sort(a[1] + 1, a[1] + 1 + n);
        int ans = 0;
```

```
        for (int i = 0; i < 2; ++i)
        {
            int t = 0, last = 0;
            memset(cnt, 0, sizeof cnt);
            memset(sum, 0, sizeof sum);
            for (int j = 1; j <= n; ++j)
            {
                int l = lower_bound(all[i] + 1, all[i] + 1 + m[i], a[i][j].l) - all[i];
                int r = lower_bound(all[i] + 1, all[i] + 1 + m[i], a[i][j].r) - all[i];
                if (l < r) update(i, l, r - 1, a[i][j].d, 1, m[i], 1);
                t += abs(sum[1] - last);
                last = sum[1];
            }
            ans += t;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

3.3 Splay Tree

```
#define key_value ch[ch[root][1]][0]
const int maxn = "Edit";
struct Splay
{
    int a[maxn];
    int sz[maxn], ch[maxn][2], fa[maxn];
    int key[maxn], rev[maxn];
    int root, tot;
    int stk[maxn], top;
    void init(int n)
    {
        tot = 0, top = 0;
        root = newnode(0, -1);
        ch[root][1] = newnode(root, -1);
        for (int i = 0; i < n; i++) a[i] = i + 1;
        key_value = build(0, n - 1, ch[root][1]);
        pushup(ch[root][1]);
        pushup(root);
    }
    int newnode(int p = 0, int k = 0)
    {
        int x = top ? stk[top--] : ++tot;
        fa[x] = p;
        sz[x] = 1;
        ch[x][0] = ch[x][1] = 0;
        key[x] = k;
        rev[x] = 0;
        return x;
    }
    void pushdown(int x)
    {
        if (rev[x])
        {
            swap(ch[x][0], ch[x][1]);
            if (ch[x][0]) rev[ch[x][0]] ^= 1;
            if (ch[x][1]) rev[ch[x][1]] ^= 1;
```

```
            rev[x] = 0;
        }
    }
    void pushup(int x) { sz[x] = sz[ch[x][0]] + sz[ch[x][1]] + 1; }
    void rotate(int x, int d)
    {
        int y = fa[x];
        pushdown(y), pushdown(x);
        ch[y][d ^ 1] = ch[x][d];
        fa[ch[x][d]] = y;
        if (fa[y]) ch[fa[y]][ch[fa[y]][1] == y] = x;
        fa[x] = fa[y];
        ch[x][d] = y;
        fa[y] = x;
        pushup(y);
    }
    void splay(int x, int goal = 0)
    {
        pushdown(x);
        while (fa[x] != goal)
        {
            if (fa[fa[x]] == goal)
                rotate(x, ch[fa[x]][0] == x);
            else
            {
                int y = fa[x];
                int d = ch[fa[y]][0] == y;
                ch[y][d] == x ? rotate(x, d ^ 1) : rotate(y, d);
                rotate(x, d);
            }
        }
        pushup(x);
        if (goal == 0) root = x;
    }
    int kth(int r, int k)
    {
        pushdown(r);
        int t = sz[ch[r][0]] + 1;
        if (t == k) return r;
        return t > k ? kth(ch[r][0], k) : kth(ch[r][1], k - t);
    }
    int build(int l, int r, int p)
    {
        if (l > r) return 0;
        int mid = l + r >> 1;
        int x = newnode(p, a[mid]);
        ch[x][0] = build(l, mid - 1, x);
        ch[x][1] = build(mid + 1, r, x);
        pushup(x);
        return x;
    }
    void select(int l, int r)
    {
        splay(kth(root, l), 0);
        splay(kth(ch[root][1], r - l + 2), root);
    }
    // 各种操作
};
```

3.4 Functional Segment Tree

静态查询区间第 k 小的值
必要时进行离散化

```
const int maxn = "Edit";
int a[maxn], rt[maxn];
int cnt;
int lson[maxn << 5], rson[maxn << 5], sum[maxn << 5];
#define Lson l, m, lson[x], lson[y]
#define Rson m + 1, r, rson[x], rson[y]
void update(int p, int l, int r, int& x, int y)
{
    lson[++cnt] = lson[y], rson[cnt] = rson[y], sum[cnt] = sum[y] + 1, x = cnt;
    if (l == r) return;
    int m = (l + r) >> 1;
    if (p <= m) update(p, Lson);
    else update(p, Rson);
}
int query(int l, int r, int x, int y, int k)
{
    if (l == r) return l;
    int m = (l + r) >> 1;
    int s = sum[lson[y]] - sum[lson[x]];
    if (s >= k) return query(Lson, k);
    else return query(Rson, k - s);
}
```

3.5 Sparse Table

```
const int maxn = "Edit";
int dp[maxn][20];
int a[maxn];
void init(int n)
{
    for (int i = 1; i <= n; i++) dp[i][0] = a[i];
    for (int j = 1; (1 << j) <= n; j++)
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
            dp[i][j] = max(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
}
// 返回[l,r]最大值
int rmq(int l, int r, int op)
{
    int k = 31 - __builtin_clz(r - l + 1);
    return max(dp[l][k], dp[r - (1 << k) + 1][k]);
}
```

二维 RMQ

```
void init(int n, int m)
{
    for (int i = 0; (1 << i) <= n; i++)
        for (int j = 0; (1 << j) <= m; j++)
        {
            if (i == 0 && j == 0) continue;
            for (int row = 1; row + (1 << i) - 1 <= n; row++)
                for (int col = 1; col + (1 << j) - 1 <= m; col++)
                    if (i)
                        dp[row][col][i][j] = max(dp[row][col][i - 1][j],
```

```
                                              dp[row + (1 << (i - 1))][col][i - 1][j]);
                        else
                            dp[row][col][i][j] = max(dp[row][col][i][j - 1],
                                                     dp[row][col + (1 << (j - 1))][i][j - 1]);
        }
}
int rmq(int x1, int y1, int x2, int y2)
{
    int kx = 31 - __builtin_clz(x2 - x1 + 1);
    int ky = 31 - __builtin_clz(y2 - y1 + 1);
    int m1 = dp[x1][y1][kx][ky];
    int m2 = dp[x2 - (1 << kx) + 1][y1][kx][ky];
    int m3 = dp[x1][y2 - (1 << ky) + 1][kx][ky];
    int m4 = dp[x2 - (1 << kx) + 1][y2 - (1 << ky) + 1][kx][ky];
    return max({m1, m2, m3, m4});
}
```

3.6 Heavy-Light Decomposition

```
const int maxn = "Edit";
struct HLD
{
    int n, dfs_clock;
    int sz[maxn], top[maxn], son[maxn], dep[maxn], fa[maxn], id[maxn];
    vector<int> G[maxn];
    // vector<pair<PII, int>> edges; 维护边权时，将其下放为儿子结点的点权
    void init(int n)
    {
        this->n = n, memset(son, -1, sizeof(son)), dfs_clock = 0;
        for (int i = 0; i <= n; i++) G[i].clear();
    }
    void add_edge(int u, int v) { G[u].push_back(v), G[v].push_back(u); }
    void dfs(int u, int p, int d)
    {
        dep[u] = d, fa[u] = p, sz[u] = 1;
        for (auto& v : G[u])
        {
            if (v == p) continue;
            dfs(v, u, d + 1);
            sz[u] += sz[v];
            if (son[u] == -1 || sz[v] > sz[son[u]]) son[u] = v;
        }
    }
    void link(int u, int t)
    {
        top[u] = t, id[u] = ++dfs_clock;
        if (son[u] == -1) return;
        link(son[u], t);
        for (auto& v : G[u])
            if (v != son[u] && v != fa[u]) link(v, v);
    }
    int query_path(int u, int v)
    { // 数据结构相关操作，一般使用线段树或者树状数组
        int ret = 0;
        while (top[u] != top[v])
        {
            if (dep[top[u]] < dep[top[v]]) swap(u, v);
            ret += query(id[top[u]], id[u]);
```

```
            u = fa[top[u]];
        }
        if (dep[u] > dep[v]) swap(u, v);
        ret += query(id[u], id[v]);
        /* 边权
        if (u == v) return ret;
        if (dep[u] > dep[v]) swap(u, v);
        ret += query(id[son[u]], id[v]);
        */
        return ret;
    }
};
```

3.7 Link-Cut Tree

动态维护一个森林

```
const int maxn = "Edit";
struct LCT
{
    int val[maxn], sum[maxn]; // 基于点权
    int rev[maxn], ch[maxn][2], fa[maxn];
    int stk[maxn];
    inline void init(int n)
    { // 初始化点权
        for (int i = 1; i <= n; i++) scanf("%d", val + i);
        for (int i = 1; i <= n; i++)
            fa[i] = ch[i][0] = ch[i][1] = rev[i] = 0;
    }
    inline bool isroot(int x) { return ch[fa[x]][0] != x && ch[fa[x]][1] != x; }
    inline bool get(int x) { return ch[fa[x]][1] == x; }
    void pushdown(int x)
    {
        if (!rev[x]) return;
        swap(ch[x][0], ch[x][1]);
        if (ch[x][0]) rev[ch[x][0]] ^= 1;
        if (ch[x][1]) rev[ch[x][1]] ^= 1;
        rev[x] ^= 1;
    }
    void pushup(int x) { sum[x] = val[x] + sum[ch[x][0]] + sum[ch[x][1]]; }
    void rotate(int x)
    {
        int y = fa[x], z = fa[fa[x]], d = get(x);
        if (!isroot(y)) ch[z][get(y)] = x;
        fa[x] = z;
        ch[y][d] = ch[x][d ^ 1], fa[ch[y][d]] = y;
        ch[x][d ^ 1] = y, fa[y] = x;
        pushup(y), pushup(x);
    }
    void splay(int x)
    {
        int top = 0;
        stk[++top] = x;
        for (int i = x; !isroot(i); i = fa[i]) stk[++top] = fa[i];
        for (int i = top; i; i--) pushdown(stk[i]);
        for (int f; !isroot(x); rotate(x))
            if (!isroot(f = fa[x])) rotate(get(x) == get(f) ? f : x);
    }
```

```
    void access(int x)
    {
        for (int y = 0; x; y = x, x = fa[x]) splay(x), ch[x][1] = y, pushup(x);
    }
    int find(int x)
    {
        access(x), splay(x);
        while (ch[x][0]) x = ch[x][0];
        return x;
    }
    void makeroot(int x) { access(x), splay(x), rev[x] ^= 1; }
    void link(int x, int y) { makeroot(x), fa[x] = y, splay(x); }
    void cut(int x, int y) { makeroot(x), access(y), splay(y), fa[x] = ch[y][0] = 0; }
    void update(int x, int v) { val[x] = v, access(x), splay(x); }
    int query(int x, int y)
    {
        makeroot(y), access(x), splay(x);
        return sum[x];
    }
};
```

3.8 Virtual Tree

```
const int maxn = "Edit";
vector<int> vtree[maxn];
void build(vector<int>& vec)
{
    sort(vec.begin(), vec.end(), [&](int x, int y) { return dfn[x] < dfn[y]; });
    static int s[maxn];
    int top = 0;
    s[top] = 0;
    vtree[0].clear();
    for (auto& u : vec)
    {
        int vlca = lca(u, s[top]);
        vtree[u].clear();
        if (vlca == s[top])
            s[++top] = u;
        else
        {
            while (top && dep[s[top - 1]] >= dep[vlca])
            {
                vtree[s[top - 1]].push_back(s[top]);
                top--;
            }
            if (s[top] != vlca)
            {
                vtree[vlca].clear();
                vtree[vlca].push_back(s[top--]);
                s[++top] = vlca;
            }
            s[++top] = u;
        }
    }
    for (int i = 0; i < top; ++i) vtree[s[i]].push_back(s[i + 1]);
}
```

3.9 Cartesian Tree

```
const int maxn = "Edit";
int lson[maxn], rson[maxn], fa[maxn];
void build(int n)
{
    stack<int> s;
    for (int i = 0; i < n; i++)
    {
        int last = -1;
        while (!s.empty() && a[i] > a[s.top()]) last = s.top(), s.pop();
        if (!s.empty()) rson[s.top()] = i, fa[i] = s.top();
        lson[i] = last;
        if (~last) fa[last] = i;
        s.push(i);
    }
}
```

4 Graph Theory

4.1 Shortest Path

```
struct Edge
{
    int from, to, dist;
    Edge() {}
    Edge(int u, int v, int d) : from(u), to(v), dist(d) {}
};
```

4.1.1 Dijkstra

```
struct HeapNode
{
    int d, u;
    bool operator<(const HeapNode& rhs) const
    {
        return d > rhs.d;
    }
};
const int maxn = "Edit";
struct Dijkstra
{
    int n, m;             // 点数和边数
    vector<Edge> edges;   // 边列表
    vector<int> G[maxn];  // 每个节点出发的边编号（从0开始编号）
    bool done[maxn];      // 是否已永久标号
    int d[maxn];          // s到各点的距离
    int p[maxn];          // 最短路中的一条边
    void init(int n)
    {
        this->n = n;
        for (int i = 0; i < n; i++) G[i].clear(); // 清空邻接表
        edges.clear();                            // 清空边表
    }
    void AddEdge(int from, int to, int dist)
    { // 如果是无向图，每条无向边需调用两次AddEdge
        edges.emplace_back(from, to, dist);
        m = edges.size();
        G[from].push_back(m - 1);
    }
    void dijkstra(int s)
    {
        priority_queue<HeapNode> q;
        for (int i = 0; i < n; i++) d[i] = INF;
        d[s] = 0;
        memset(done, 0, sizeof(done));
        q.push({0, s});
        while (!q.empty())
        {
            HeapNode x = q.top();
            q.pop();
            int u = x.u;
            if (done[u]) continue;
            done[u] = true;
            for (auto& id : G[u])
            {
```

```
                Edge& e = edges[id];
                if (d[e.to] > d[u] + e.dist)
                {
                    d[e.to] = d[u] + e.dist;
                    p[e.to] = id;
                    q.push({d[e.to], e.to});
                }
            }
        }
    }
};
```

4.1.2 Bellman-Ford

```
const int maxn = "Edit";
struct BellmanFord
{
    int n, m;
    vector<Edge> edges;
    vector<int> G[maxn];
    bool inq[maxn]; // 是否在队列中
    int d[maxn];    // s到各个点的距离
    int p[maxn];    // 最短路中的上一条弧
    int cnt[maxn];  // 进队次数
    void init(int n)
    {
        this->n = n;
        for (int i = 0; i < n; i++) G[i].clear();
        edges.clear();
    }
    void AddEdge(int from, int to, int dist)
    {
        edges.emplace_back(from, to, dist);
        m = edges.size();
        G[from].push_back(m - 1);
    }
    bool bellmanford(int s)
    {
        queue<int> q;
        memset(inq, 0, sizeof(inq));
        memset(cnt, 0, sizeof(cnt));
        for (int i = 0; i < n; i++) d[i] = INF;
        d[s] = 0;
        inq[s] = true;
        q.push(s);
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            inq[u] = false;
            for (auto& id : G[u])
            {
                Edge& e = edges[id];
                if (d[u] < INF && d[e.to] > d[u] + e.dist)
                {
                    d[e.to] = d[u] + e.dist;
                    p[e.to] = id;
                    if (!inq[e.to])
```

```
                    {
                        q.push(e.to);
                        inq[e.to] = true;
                        if (++cnt[e.to] > n) return false;
                    }
                }
            }
        }
        return true;
    }
};
```

4.2 Minimal Spanning Tree

4.2.1 Zhu Liu

```
const int maxn = "Edit";
// 固定根的最小树型图，邻接矩阵写法
struct MDST
{
    int n;
    int w[maxn][maxn]; // 边权
    int vis[maxn];     // 访问标记，仅用来判断无解
    int ans;           // 计算答案
    int removed[maxn]; // 每个点是否被删除
    int cid[maxn];     // 所在圈编号
    int pre[maxn];     // 最小入边的起点
    int iw[maxn];      // 最小入边的权值
    int max_cid;       // 最大圈编号
    void init(int n)
    {
        this->n = n;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) w[i][j] = INF;
    }
    void AddEdge(int u, int v, int cost)
    {
        w[u][v] = min(w[u][v], cost); // 重边取权最小的
    }
    // 从s出发能到达多少个结点
    int dfs(int s)
    {
        vis[s] = 1;
        int ans = 1;
        for (int i = 0; i < n; i++)
            if (!vis[i] && w[s][i] < INF) ans += dfs(i);
        return ans;
    }
    // 从u出发沿着pre指针找圈
    bool cycle(int u)
    {
        max_cid++;
        int v = u;
        while (cid[v] != max_cid)
        {
            cid[v] = max_cid;
            v = pre[v];
        }
        return v == u;
```

```
    }
    // 计算u的最小入弧，入弧起点不得在圈c中
    void update(int u)
    {
        iw[u] = INF;
        for (int i = 0; i < n; i++)
            if (!removed[i] && w[i][u] < iw[u])
            {
                iw[u] = w[i][u];
                pre[u] = i;
            }
    }
    // 根结点为s，如果失败则返回false
    bool solve(int s)
    {
        memset(vis, 0, sizeof(vis));
        if (dfs(s) != n) return false;
        memset(removed, 0, sizeof(removed));
        memset(cid, 0, sizeof(cid));
        for (int u = 0; u < n; u++) update(u);
        pre[s] = s;
        iw[s] = 0; // 根结点特殊处理
        ans = max_cid = 0;
        for (;;)
        {
            bool have_cycle = false;
            for (int u = 0; u < n; u++)
                if (u != s && !removed[u] && cycle(u))
                {
                    have_cycle = true;
                    // 以下代码缩圈，圈上除了u之外的结点均删除
                    int v = u;
                    do
                    {
                        if (v != u) removed[v] = 1;
                        ans += iw[v];
                        // 对于圈外点i，把边i->v改成i->u（并调整权值）；v->i改为u->i
                        // 注意圈上可能还有一个v'使得i->v'或者v'->i存在，
                        // 因此只保留权值最小的i->u和u->i
                        for (int i = 0; i < n; i++)
                            if (cid[i] != cid[u] && !removed[i])
                            {
                                if (w[i][v] < INF)
                                    w[i][u] = min(w[i][u], w[i][v] - iw[v]);
                                w[u][i] = min(w[u][i], w[v][i]);
                                if (pre[i] == v) pre[i] = u;
                            }
                        v = pre[v];
                    } while (v != u);
                    update(u);
                    break;
                }
            if (!have_cycle) break;
        }
        for (int i = 0; i < n; i++)
            if (!removed[i]) ans += iw[i];
        return true;
    }
};
```

4.3 LCA

4.3.1 DFS+ST

DFS+ST 在线算法
时间复杂度 $O(nlogn + q)$

```
const int maxn = "Edit";
vector<int> G[maxn], sp;
int dep[maxn], dfn[maxn];
PII dp[21][maxn << 1];
void init(int n)
{
    for (int i = 0; i < n; i++) G[i].clear();
    sp.clear();
}
void dfs(int u, int fa)
{
    dep[u] = dep[fa] + 1;
    dfn[u] = sp.size();
    sp.push_back(u);
    for (auto& v : G[u])
    {
        if (v == fa) continue;
        dfs(v, u);
        sp.push_back(u);
    }
}
void initrmq()
{
    int n = sp.size();
    for (int i = 0; i < n; i++) dp[0][i] = {dfn[sp[i]], sp[i]};
    for (int i = 1; (1 << i) <= n; i++)
        for (int j = 0; j + (1 << i) - 1 < n; j++)
            dp[i][j] = min(dp[i - 1][j], dp[i - 1][j + (1 << (i - 1))]);
}
int lca(int u, int v)
{
    int l = dfn[u], r = dfn[v];
    if (l > r) swap(l, r);
    int k = 31 - __builtin_clz(r - l + 1);
    return min(dp[k][l], dp[k][r - (1 << k) + 1]).second;
}
```

4.3.2 Tarjan

Tarjan 离线算法
时间复杂度 $O(n + q)$

```
const int maxn = "Edit";
int par[maxn];              //并查集
int ans[maxn];              //存储答案
vector<int> G[maxn];        //邻接表
vector<PII> query[maxn]; //存储查询信息
bool vis[maxn];             //是否被遍历
inline void init(int n)
{
    for (int i = 1; i <= n; i++)
    {
```

```cpp
        G[i].clear(), query[i].clear();
        par[i] = i, vis[i] = 0;
    }
}
inline void add_edge(int u, int v) { G[u].push_back(v); }
inline void add_query(int id, int u, int v)
{
    query[u].emplace_back(v, id);
    query[v].emplace_back(u, id);
}
void tarjan(int u)
{
    vis[u] = 1;
    for (auto& v : G[u])
    {
        if (vis[v]) continue;
        tarjan(v);
        unite(u, v);
    }
    for (auto& q : query[u])
    {
        int &v = q.X, &id = q.Y;
        if (!vis[v]) continue;
        ans[id] = find(v);
    }
}
```

4.4 Depth-First Traversal

4.4.1 Biconnected-Component

```cpp
//割顶的bccno无意义
const int maxn = "Edit";
int pre[maxn], iscut[maxn], bccno[maxn], dfs_clock, bcc_cnt;
vector<int> G[maxn], bcc[maxn];
stack<PII> s;
void init(int n)
{
    for (int i = 0; i < n; i++) G[i].clear();
}
inline void add_edge(int u, int v) { G[u].push_back(v), G[v].push_back(u); }
int dfs(int u, int fa)
{
    int lowu = pre[u] = ++dfs_clock;
    int child = 0;
    for (auto& v : G[u])
    {
        PII e = {u, v};
        if (!pre[v])
        {
            //没有访问过v
            s.push(e);
            child++;
            int lowv = dfs(v, u);
            lowu = min(lowu, lowv); //用后代的low函数更新自己
            if (lowv >= pre[u])
            {
                iscut[u] = true;
                bcc_cnt++;
```

```
                bcc[bcc_cnt].clear(); //注意! bcc从1开始编号
                for (;;)
                {
                    PII x = s.top();
                    s.pop();
                    if (bccno[x.first] != bcc_cnt)
                        bcc[bcc_cnt].push_back(x.first), bcc[x.first] = bcc_cnt;
                    if (bccno[x.second] != bcc_cnt)
                        bcc[bcc_cnt].push_back(x.second), bcc[x.second] = bcc_cnt;
                    if (x.first == u && x.second == v) break;
                }
            }
        }
        else if (pre[v] < pre[u] && v != fa)
        {
            s.push(e);
            lowu = min(lowu, pre[v]); //用反向边更新自己
        }
    }
    if (fa < 0 && child == 1) iscut[u] = 0;
    return lowu;
}
void find_bcc(int n)
{
    //调用结束后s保证为空，所以不用清空
    memset(pre, 0, sizeof(pre));
    memset(iscut, 0, sizeof(iscut));
    memset(bccno, 0, sizeof(bccno));
    dfs_clock = bcc_cnt = 0;
    for (int i = 0; i < n; i++)
        if (!pre[i]) dfs(i, -1);
}
```

4.4.2 Strongly Connected Component

```
const int maxn = "Edit";
vector<int> G[maxn];
int pre[maxn], lowlink[maxn], sccno[maxn], dfs_clock, scc_cnt;
stack<int> S;
inline void init(int n)
{
    for (int i = 0; i < n; i++) G[i].clear();
}
inline void add_edge(int u, int v) { G[u].push_back(v); }
void dfs(int u)
{
    pre[u] = lowlink[u] = ++dfs_clock;
    S.push(u);
    for (auto& v : G[u])
    {
        if (!pre[v])
        {
            dfs(v);
            lowlink[u] = min(lowlink[u], lowlink[v]);
        }
        else if (!sccno[v])
            lowlink[u] = min(lowlink[u], pre[v]);
    }
```

```
    if (lowlink[u] == pre[u])
    {
        scc_cnt++;
        for (;;)
        {
            int x = S.top();
            S.pop();
            sccno[x] = scc_cnt;
            if (x == u) break;
        }
    }
}
void find_scc(int n)
{
    dfs_clock = 0, scc_cnt = 0;
    memset(sccno, 0, sizeof(sccno)), memset(pre, 0, sizeof(pre));
    for (int i = 0; i < n; i++)
        if (!pre[i]) dfs(i);
}
```

4.4.3 2-SAT

```
const int maxn = "Edit";
struct TwoSAT
{
    int n;
    vector<int> G[maxn << 1];
    bool mark[maxn << 1];
    int S[maxn << 1], c;
    void init(int n)
    {
        this->n = n;
        for (int i = 0; i < (n << 1); i++) G[i].clear();
        memset(mark, 0, sizeof(mark));
    }
    bool dfs(int x)
    {
        if (mark[x ^ 1]) return false;
        if (mark[x]) return true;
        mark[x] = true;
        S[c++] = x;
        for (auto& y : G[x])
            if (!dfs(y)) return false;
        return true;
    }
    //x = xval or y = yval
    void add_clause(int x, int xval, int y, int yval)
    {
        x = (x << 1) + xval;
        y = (y << 1) + yval;
        G[x ^ 1].push_back(y);
        G[y ^ 1].push_back(x);
    }
    bool solve()
    {
        for (int i = 0; i < (n << 1); i += 2)
            if (!mark[i] && !mark[i + 1])
            {
```

```
                    c = 0;
                    if (!dfs(i))
                    {
                        while (c > 0) mark[S[--c]] = false;
                        if (!dfs(i + 1)) return false;
                    }
                }
            return true;
        }
};
```

4.5 Eular Path

- 基本概念:
  - 欧拉图: 能够没有重复地一次遍历所有边的图。(必须是连通图)
  - 欧拉路: 上述遍历的路径就是欧拉路。
  - 欧拉回路: 若欧拉路是闭合的（一个圈，从起点开始遍历最终又回到起点），则为欧拉回路。
- 无向图 G 有欧拉路径的充要条件
  - G 是连通图
  - G 中奇顶点（连接边的数量为奇数）的数量等于 0 或 2.
- 无向图 G 有欧拉回路的充要条件
  - G 是连通图
  - G 中每个顶点都是偶顶点
- 有向图 G 有欧拉路径的充要条件
  - G 是连通图
  - u 的出度比入度大 1，v 的出度比入度小 1，其他所有点出度和入度相同。(u 为起点，v 为终点)
- 有向图 G 有欧拉回路的充要条件
  - G 是连通图
  - G 中每个顶点的出度等于入度

4.5.1 Fleury

若有两个点的度数是奇数，则此时这两个点只能作为欧拉路径的起点和终点。

```
const int maxn = "Edit";
int G[maxn][maxn];
int deg[maxn][maxn];
vector<int> ans;
inline void init() { memset(G, 0, sizeof(G)), memset(deg, 0, sizeof(deg)); }
inline void AddEdge(int u, int v) { deg[u]++, deg[v]++, G[u][v]++, G[v][u]++; }
void Fleury(int s)
{
    for (int i = 0; i < n; i++)
        if (G[s][i])
        {
            G[s][i]--, G[i][s]--;
            Fleury(i);
        }
    ans.push_back(s);
}
```

4.6 Bipartite Graph Matching

1. 一个二分图中的最大匹配数等于这个图中的最小点覆盖数
2. 最小路径覆盖 $=|G|$-最大匹配数

   在一个 $N \times N$ 的有向图中, 路径覆盖就是在图中找一些路经, 使之覆盖了图中的所有顶点, 且任何一个顶点有且只有一条路径与之关联;

   (如果把这些路径中的每条路径从它的起始点走到它的终点, 那么恰好可以经过图中的每个顶点一次且仅一次); 如果不考虑图中存在回路, 那么每每条路径就是一个弱连通子集.

   由上面可以得出:

   (a) 一个单独的顶点是一条路径;

   (b) 如果存在一路径 $p_1, p_2, ...p_k$, 其中 p_1 为起点,p_k 为终点, 那么在覆盖图中, 顶点 $p_1, p2, ...p_k$ 不再与其它的顶点之间存在有向边.

   最小路径覆盖就是找出最小的路径条数, 使之成为 G 的一个路径覆盖.

   路径覆盖与二分图匹配的关系: 最小路径覆盖 $=|G|$-最大匹配数;
3. 二分图最大独立集 = 顶点数-二分图最大匹配

   独立集: 图中任意两个顶点都不相连的顶点集合。

4.6.1 Hungry(Matrix)

时间复杂度:$O(VE)$.
顶点编号从 0 开始

```
const int maxn = "Edit";
int uN, vN;         //uN是匹配左边的顶点数,vN是匹配右边的顶点数
int g[maxn][maxn]; //邻接矩阵g[i][j]表示i->j的有向边就可以了,是左边向右边的匹配
int linker[maxn];
bool used[maxn];
bool dfs(int u)
{
    for (int v = 0; v < vN; v++)
        if (g[u][v] && !used[v])
        {
            used[v] = true;
            if (linker[v] == -1 || dfs(linker[v]))
            {
                linker[v] = u;
                return true;
            }
        }
    return false;
}
int hungary()
{
    int res = 0;
    memset(linker, -1, sizeof(linker));
    for (int u = 0; u < uN; u++)
    {
        memset(used, 0, sizeof(used));
        if (dfs(u)) res++;
    }
    return res;
}
```

4.6.2 Hungry(List)

使用前用 init() 进行初始化
加边使用函数 addedge(u,v)

```
const int maxn = "Edit";
int n;
vector<int> G[maxn];
int linker[maxn];
bool used[maxn];
inline void init(int n)
{
    for (int i = 0; i < n; i++) G[i].clear();
}
inline void addedge(int u, int v) { G[u].push_back(v); }
bool dfs(int u)
{
    for (auto& v : G[u])
    {
        if (!used[v])
        {
            used[v] = true;
            if (linker[v] == -1 || dfs(linker[v]))
            {
                linker[v] = u;
                return true;
            }
        }
    }
    return false;
}
int hungary()
{
    int ans = 0;
    memset(linker, -1, sizeof(linker));
    for (int u = 0; u < n; u++)
    {
        memset(used, 0, sizeof(used));
        if (dfs(u)) ans++;
    }
    return ans;
}
```

4.6.3 Hopcroft-Carp

复杂度 $O(\sqrt{n} * E)$
uN 为左端的顶点数, 使用前赋值 (点编号 0 开始)

```
const int maxn = "Edit";
vector<int> G[maxn];
int uN, dis;
int Mx[maxn], My[maxn];
int dx[maxn], dy[maxn];
bool used[maxn];
inline void init(int n)
{
    for (int i = 0; i < n; i++) G[i].clear();
}
```

```cpp
inline void addedge(int u, int v) { G[u].push_back(v); }
bool bfs()
{
    queue<int> q;
    dis = INF;
    memset(dx, -1, sizeof(dx)), memset(dy, -1, sizeof(dy));
    for (int i = 0; i < uN; i++)
        if (Mx[i] == -1) q.push(i), dx[i] = 0;
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        if (dx[u] > dis) break;
        for (auto& v : G[u])
        {
            if (dy[v] == -1)
            {
                dy[v] = dx[u] + 1;
                if (My[v] == -1)
                    dis = dy[v];
                else
                {
                    dx[My[v]] = dy[v] + 1;
                    q.push(My[v]);
                }
            }
        }
    }
    return dis != INF;
}
bool dfs(int u)
{
    for (auto& v : G[u])
    {
        if (!used[v] && dy[v] == dx[u] + 1)
        {
            used[v] = true;
            if (My[v] != -1 && dy[v] == dis) continue;
            if (My[v] == -1 || dfs(My[v]))
            {
                My[v] = u, Mx[u] = v;
                return true;
            }
        }
    }
    return false;
}
int MaxMatch()
{
    int res = 0;
    memset(Mx, -1, sizeof(Mx)), memset(My, -1, sizeof(My));
    while (bfs())
    {
        memset(used, false, sizeof(used));
        for (int i = 0; i < uN; i++)
            if (Mx[i] == -1 && dfs(i)) res++;
    }
    return res;
}
```

4.6.4 Hungry(Multiple)

```
const int maxn = "Edit";
const int maxm = "Edit";
int uN, vN;         //u,v的数目,使用前面必须赋值
int g[maxn][maxm]; //邻接矩阵
int linker[maxm][maxn];
bool used[maxm];
int num[maxm]; //右边最大的匹配数
bool dfs(int u)
{
    for (int v = 0; v < vN; v++)
        if (g[u][v] && !used[v])
        {
            used[v] = true;
            if (linker[v][0] < num[v])
            {
                linker[v][++linker[v][0]] = u;
                return true;
            }
            for (int i = 1; i <= num[0]; i++)
                if (dfs(linker[v][i]))
                {
                    linker[v][i] = u;
                    return true;
                }
        }
    return false;
}
int hungary()
{
    int res = 0;
    for (int i = 0; i < vN; i++) linker[i][0] = 0;
    for (int u = 0; u < uN; u++)
    {
        memset(used, 0, sizeof(used));
        if (dfs(u)) res++;
    }
    return res;
}
```

4.6.5 Kuhn-Munkres

```
const int maxn = "Edit";
int n;
int cost[maxn][maxn];
int lx[maxn], ly[maxn], match[maxn], slack[maxn];
int prev[maxn];
bool vy[maxn];

void augment(int root)
{
    fill(vy + 1, vy + n + 1, false);
    fill(slack + 1, slack + n + 1, INF);
    int py;
    match[py = 0] = root;
    do
    {
```

```
            vy[py] = true;
            int x = match[py], yy;
            int delta = INF;
            for (int y = 1; y <= n; y++)
            {
                if (!vy[y])
                {
                    if (lx[x] + ly[y] - cost[x][y] < slack[y])
                        slack[y] = lx[x] + ly[y] - cost[x][y], prev[y] = py;
                    if (slack[y] < delta) delta = slack[y], yy = y;
                }
            }
            for (int y = 0; y <= n; y++)
            {
                if (vy[y])
                    lx[match[y]] -= delta, ly[y] += delta;
                else
                    slack[y] -= delta;
            }
            py = yy;
        } while (match[py] != -1);
        do
        {
            int pre = prev[py];
            match[py] = match[pre], py = pre;
        } while (py);
}
int KM()
{
    for (int i = 1; i <= n; i++)
    {
        lx[i] = ly[i] = 0;
        match[i] = -1;
        for (int j = 1; j <= n; j++) lx[i] = max(lx[i], cost[i][j]);
    }
    int answer = 0;
    for (int root = 1; root <= n; root++) augment(root);
    for (int i = 1; i <= n; i++) answer += lx[i], answer += ly[i];
    return answer;
}
```

4.7 Network Flow

```
struct Edge
{
    int from, to, cap, flow;
    Edge(int u, int v, int c, int f)
        : from(u), to(v), cap(c), flow(f) {}
};
```

费用流

```
struct Edge
{
    int from, to, cap, flow, cost;
    Edge(int u, int v, int c, int f, int w)
        : from(u), to(v), cap(c), flow(f), cost(w) {}
};
```

建模技巧

二分图带权最大独立集。给出一个二分图，每个结点上有一个正权值。要求选出一些点，使得这些点之间没有边相连，且权值和最大。

解：在二分图的基础上添加源点 S 和汇点 T，然后从 S 向所有 X 集合中的点连一条边，所有 Y 集合中的点向 T 连一条边，容量均为该点的权值。X 结点与 Y 结点之间的边的容量均为无穷大。这样，对于图中的任意一个割，将割中的边对应的结点删掉就是一个符合要求的解，权和为所有权减去割的容量。因此，只需要求出最小割，就能求出最大权和。

公平分配问题。把 m 个任务分配给 n 个处理器。其中每个任务有两个候选处理器，可以任选一个分配。要求所有处理器中，任务数最多的那个处理器所分配的任务数尽量少。不同任务的候选处理器集 $\{p_1, p_2\}$ 保证不同。

解：本题有一个比较明显的二分图模型，即 X 结点是任务，Y 结点是处理器。二分答案 x，然后构图，首先从源点 S 出发向所有的任务结点引一条边，容量等于 1，然后从每个任务结点出发引两条边，分别到达它所能分配到的两个处理器结点，容量为 1，最后从每个处理器结点出发引一条边到汇点 T，容量为 x，表示选择该处理器的任务不能超过 x。这样网络中的每个单位流量都是从 S 流到一个任务结点，再到处理器结点，最后到汇点 T。只有当网络中的总流量等于 m 时才意味着所有任务都选择了一个处理器。这样，我们通过 $O(\log m)$ 次最大流便算出了答案。

区间 k 覆盖问题。数轴上有一些带权值的左闭右开区间。选出权和尽量大的一些区间，使得任意一个数最多被 k 个区间覆盖。

解：本题可以用最小费用流解决，构图方法是把每个数作为一个结点，然后对于权值为 w 的区间 $[u, v)$ 加边 $u \to v$，容量为 1，费用为 $-w$。再对所有相邻的点加边 $i \to i+1$，容量为 k，费用为 0。最后，求最左点到最右点的最小费用最大流即可，其中每个流量对应一组互不相交的区间。如果数值范围太大，可以先进行离散化。

最大闭合子图。给定带权图 G（权值可正可负），求一个权和最大的点集，使得起点在该点集中的任意弧，终点也在该点集中。

解：新增附加源 s 和附加汇 t，从 s 向所有正权点引一条边，容量为权值；从所有负权点向汇点引一条边，容量为权值的相反数。求出最小割以后，$S - \{s\}$ 就是最大闭合子图。

最大密度子图。给出一个无向图，找一个点集，使得这些点之间的边数除以点数的值（称为子图的密度）最大。

解：如果两个端点都选了，就必然要选边，这就是一种推导。如果把每个点和每条边都看成新图中的结点，可以把问题转化为最大闭合子图。

4.7.1 EdmondKarp

```
const int maxn = "Edit";
struct EdmonsKarp //时间复杂度O(v*E*E)
{
    int n, m;
    vector<Edge> edges;  //边数的两倍
    vector<int> G[maxn]; //邻接表，G[i][j]表示节点i的第j条边在e数组中的序号
    int a[maxn];         //起点到i的可改进量
    int p[maxn];         //最短路树上p的入弧编号
    void init(int n)
    {
        for (int i = 0; i < n; i++) G[i].clear();
        edges.clear();
    }
    void AddEdge(int from, int to, int cap)
    {
        edges.emplace_back(from, to, cap, 0);
        edges.emplace_back(to, from, 0, 0); //反向弧
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }
```

```
    int Maxflow(int s, int t)
    {
        int flow = 0;
        for (;;)
        {
            memset(a, 0, sizeof(a));
            queue<int> q;
            q.push(s);
            a[s] = INF;
            while (!q.empty())
            {
                int x = q.front();
                q.pop();
                for (int i = 0; i < G[x].size(); i++)
                {
                    Edge& e = edges[G[x][i]];
                    if (!a[e.to] && e.cap > e.flow)
                    {
                        p[e.to] = G[x][i];
                        a[e.to] = min(a[x], e.cap - e.flow);
                        q.push(e.to);
                    }
                }
                if (a[t]) break;
            }
            if (!a[t]) break;
            for (int u = t; u != s; u = edges[p[u]].from)
            {
                edges[p[u]].flow += a[t];
                edges[p[u] ^ 1].flow -= a[t];
            }
            flow += a[t];
        }
        return flow;
    }
};
```

4.7.2 Dinic

```
const int maxn = "Edit";
struct Dinic
{
    int n, m, s, t;       //结点数，边数（包括反向弧），源点编号和汇点编号
    vector<Edge> edges;   //边表。edge[e]和edge[e^1]互为反向弧
    vector<int> G[maxn];  //邻接表，G[i][j]表示节点i的第j条边在e数组中的序号
    bool vis[maxn];       //BFS使用
    int d[maxn];          //从起点到i的距离
    int cur[maxn];        //当前弧下标
    void init(int n)
    {
        this->n = n;
        for (int i = 0; i < n; i++) G[i].clear();
        edges.clear();
    }
    void AddEdge(int from, int to, int cap)
    {
        edges.emplace_back(from, to, cap, 0);
        edges.emplace_back(to, from, 0, 0);
```

```
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }
    bool BFS()
    {
        memset(vis, 0, sizeof(vis));
        memset(d, 0, sizeof(d));
        queue<int> q;
        q.push(s);
        d[s] = 0;
        vis[s] = 1;
        while (!q.empty())
        {
            int x = q.front();
            q.pop();
            for (int i = 0; i < G[x].size(); i++)
            {
                Edge& e = edges[G[x][i]];
                if (!vis[e.to] && e.cap > e.flow)
                {
                    vis[e.to] = 1;
                    d[e.to] = d[x] + 1;
                    q.push(e.to);
                }
            }
        }
        return vis[t];
    }
    int DFS(int x, int a)
    {
        if (x == t || a == 0) return a;
        int flow = 0, f;
        for (int& i = cur[x]; i < G[x].size(); i++)
        { //从上次考虑的弧
            Edge& e = edges[G[x][i]];
            if (d[x] + 1 == d[e.to] && (f = DFS(e.to, min(a, e.cap - e.flow))) > 0)
            {
                e.flow += f;
                edges[G[x][i] ^ 1].flow -= f;
                flow += f;
                a -= f;
                if (a == 0) break;
            }
        }
        return flow;
    }
    int Maxflow(int s, int t)
    {
        this->s = s, this->t = t;
        int flow = 0;
        while (BFS())
        {
            memset(cur, 0, sizeof(cur));
            flow += DFS(s, INF);
        }
        return flow;
    }
};
```

4.7.3 ISAP

```
const int maxn = "Edit";
struct ISAP
{
    int n, m, s, t;       //结点数，边数（包括反向弧），源点编号和汇点编号
    vector<Edge> edges;  //边表。edges[e]和edges[e^1]互为反向弧
    vector<int> G[maxn]; //邻接表，G[i][j]表示结点i的第j条边在e数组中的序号
    bool vis[maxn];       //BFS使用
    int d[maxn];          //起点到i的距离
    int cur[maxn];        //当前弧下标
    int p[maxn];          //可增广路上的一条弧
    int num[maxn];        //距离标号计数
    void init(int n)
    {
        this->n = n;
        for (int i = 0; i < n; i++) G[i].clear();
        edges.clear();
    }
    void AddEdge(int from, int to, int cap)
    {
        edges.emplace_back(from, to, cap, 0);
        edges.emplace_back(to, from, 0, 0);
        int m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }
    int Augumemt()
    {
        int x = t, a = INF;
        while (x != s)
        {
            Edge& e = edges[p[x]];
            a = min(a, e.cap - e.flow);
            x = edges[p[x]].from;
        }
        x = t;
        while (x != s)
        {
            edges[p[x]].flow += a;
            edges[p[x] ^ 1].flow -= a;
            x = edges[p[x]].from;
        }
        return a;
    }
    void BFS()
    {
        memset(vis, 0, sizeof(vis));
        memset(d, 0, sizeof(d));
        queue<int> q;
        q.push(t);
        d[t] = 0;
        vis[t] = 1;
        while (!q.empty())
        {
            int x = q.front();
            q.pop();
            int len = G[x].size();
            for (int i = 0; i < len; i++)
```

```
                {
                    Edge& e = edges[G[x][i] ^ 1];
                    if (!vis[e.from] && e.cap > e.flow)
                    {
                        vis[e.from] = 1;
                        d[e.from] = d[x] + 1;
                        q.push(e.from);
                    }
                }
            }
        }
        int Maxflow(int s, int t)
        {
            this->s = s;
            this->t = t;
            int flow = 0;
            BFS();
            memset(num, 0, sizeof(num));
            for (int i = 0; i < n; i++)
                if (d[i] < INF) num[d[i]]++;
            int x = s;
            memset(cur, 0);
            while (d[s] < n)
            {
                if (x == t)
                {
                    flow += Augumemt();
                    x = s;
                }
                int ok = 0;
                for (int i = cur[x]; i < G[x].size(); i++)
                {
                    Edge& e = edges[G[x][i]];
                    if (e.cap > e.flow && d[x] == d[e.to] + 1)
                    {
                        ok = 1;
                        p[e.to] = G[x][i];
                        cur[x] = i;
                        x = e.to;
                        break;
                    }
                }
                if (!ok) //Retreat
                {
                    int m = n - 1;
                    for (int i = 0; i < G[x].size(); i++)
                    {
                        Edge& e = edges[G[x][i]];
                        if (e.cap > e.flow) m = min(m, d[e.to]);
                    }
                    if (--num[d[x]] == 0) break; //gap优化
                    num[d[x] = m + 1]++;
                    cur[x] = 0;
                    if (x != s) x = edges[p[x]].from;
                }
            }
            return flow;
        }
};
```

4.7.4 MinCost MaxFlow

```
const int maxn = "Edit";
struct MCMF
{
    int n, m;
    vector<Edge> edges;
    vector<int> G[maxn];
    int inq[maxn]; //是否在队列中
    int d[maxn];   //bellmanford
    int p[maxn];   //上一条弧
    int a[maxn];   //可改进量
    void init(int n)
    {
        this->n = n;
        for (int i = 0; i < n; i++) G[i].clear();
        edges.clear();
    }
    void AddEdge(int from, int to, int cap, int cost)
    {
        edges.emplace_back(from, to, cap, 0, cost);
        edges.emplace_back(to, from, 0, 0, -cost);
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }
    bool BellmanFord(int s, int t, int& flow, ll& cost)
    {
        for (int i = 0; i < n; i++) d[i] = INF;
        memset(inq, 0, sizeof(inq));
        d[s] = 0;
        inq[s] = 1;
        p[s] = 0;
        a[s] = INF;
        queue<int> q;
        q.push(s);
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            inq[u] = 0;
            for (int i = 0; i < G[u].size(); i++)
            {
                Edge& e = edges[G[u][i]];
                if (e.cap > e.flow && d[e.to] > d[u] + e.cost)
                {
                    d[e.to] = d[u] + e.cost;
                    p[e.to] = G[u][i];
                    a[e.to] = min(a[u], e.cap - e.flow);
                    if (!inq[e.to])
                    {
                        q.push(e.to);
                        inq[e.to] = 1;
                    }
                }
            }
        }
        if (d[t] == INF) return false; // 当没有可增广的路时退出
        flow += a[t];
```

```
        cost += (ll)d[t] * (ll)a[t];
        for (int u = t; u != s; u = edges[p[u]].from)
        {
            edges[p[u]].flow += a[t];
            edges[p[u] ^ 1].flow -= a[t];
        }
        return true;
    }
    int MincostMaxflow(int s, int t, ll& cost)
    {
        int flow = 0;
        cost = 0;
        while (BellmanFord(s, t, flow, cost));
        return flow;
    }
};
```

4.7.5 Upper-Lower Bound

上下界网络流建图方法

记号说明

- $f(u,v)$ 表示 $u \to v$ 的实际流量
- $b(u,v)$ 表示 $u \to v$ 的流量下界
- $c(u,v)$ 表示 $u \to v$ 的流量上界

无源汇可行流

建图

- 新建附加源点 S 和 T
- 原图中的边 $u \to v$，限制为 $[b,c]$，建边 $u \to v$，容量为 $c-b$
- 记 $d(i) = \sum b(u,i) - \sum b(i,v)$
- 若 $d(i) > 0$，建边 $S \to i$，流量为 $d(i)$
- 若 $d(i) < 0$，建边 $i \to T$，流量为 $-d(i)$

求解

- 跑 $S \to T$ 的最大流，如果满流，则原图存在可行流。
- 此时，原图中每一条边的流量为新图中对应边的流量加上这条边的下界。

有源汇可行流

建图

- 在原图中建边 $t \to s$，流量限制为 $[0, +\infty)$，这样就改造成了无源汇的网络流图。
- 之后就可以像求解无源汇可行流一样建图了。

求解 同无源汇可行流

有源汇最大流

建图 同有源汇可行流

求解

- 先跑一遍 $S \to T$ 的最大流，求出可行流
- 记此时 $\sum f(s,i) = sum_1$
- 将 $t \to s$ 这条边拆掉，在新图上跑 $s \to t$ 的最大流
- 记此时 $\sum f(s,i) = sum_2$
- 最终答案即为 $sum_1 + sum_2$

有源汇最小流

建图 同无源汇可行流

求解

- 求 $S \to T$ 最大流
- 建边 $t \to s$，容量为 $+\infty$
- 再跑一遍 $S \to T$ 的最大流，答案即为 $f(t,s)$

有源汇的最大流和最小流也可以通过二分答案求得，即二分 $t \to s$ 的下界（最大流）和上界（最小流）复杂度多了个 $O(\log n)$ 这里不再赘述。

蓝书上的做法

- 先用无源汇可行流建图的方法求出可行流，然后用传统 $s-t$ 增广路算法即可得到最大流。把 t 看成源点，s 看成汇点后求出的 $t-s$ 最大流就是最小流。
- 注意：原先每条弧 $u \to v$ 的反向弧容量为 0，而在有容量下界的情形中，反向弧的容量应该等于流量下界。

有源汇费用流

建图

- 新建附加源点 S 和 T
- 原图中的边 $u \to v$，限制为 $[b,c]$，费用为 $cost$，建边 $u \to v$，容量为 $c-b$，费用为 $cost$
- 记 $d(i) = \sum b(u,i) - \sum b(i,v)$
- 若 $d(i) > 0$，建边 $S \to i$，流量为 $d(i)$，**费用为** 0
- 若 $d(i) < 0$，建边 $i \to T$，流量为 $-d(i)$，**费用为** 0
- 建边 $t \to s$，流量为 $+\infty$，费用为 0。

求解

- 跑 $S \to T$ 的最小费用最大流
- 答案为求出的费用加上原图中边的下界乘以边的费用

5 Computational Geometry

5.1 Basic Function

```
#define zero(x) ((fabs(x) < eps ? 1 : 0))
#define sgn(x) (fabs(x) < eps ? 0 : ((x) < 0 ? -1 : 1))

struct point
{
    double x, y;
    point(double a = 0, double b = 0) { x = a, y = b; }
    point operator-(const point& b) const { return point(x - b.x, y - b.y); }
    point operator+(const point& b) const { return point(x + b.x, y + b.y); }
    // 两点是否重合
    bool operator==(point& b) { return zero(x - b.x) && zero(y - b.y); }
    // 点积(以原点为基准)
    double operator*(const point& b) const { return x * b.x + y * b.y; }
    // 叉积(以原点为基准)
    double operator^(const point& b) const { return x * b.y - y * b.x; }
    // 绕P点逆时针旋转a弧度后的点
    point rotate(point b, double a)
    {
        double dx, dy;
        (*this - b).split(dx, dy);
        double tx = dx * cos(a) - dy * sin(a);
        double ty = dx * sin(a) + dy * cos(a);
        return point(tx, ty) + b;
    }
    // 点坐标分别赋值到a和b
    void split(double& a, double& b) { a = x, b = y; }
};
struct line
{
    point s, e;
    line() {}
    line(point ss, point ee) { s = ss, e = ee; }
};
```

5.2 Position

5.2.1 Point-Point

```
double dist(point a, point b) { return sqrt((a - b) * (a - b)); }
```

5.2.2 Line-Line

```
// <0, *> 表示重合; <1, *> 表示平行; <2, P> 表示交点是P;
pair<int, point> spoint(line l1, line l2)
{
    point res = l1.s;
    if (sgn((l1.s - l1.e) ^ (l2.s - l2.e)) == 0)
        return {sgn((l1.s - l2.e) ^ (l2.s - l2.e)) != 0, res};
    double t = ((l1.s - l2.s) ^ (l2.s - l2.e)) / ((l1.s - l1.e) ^ (l2.s - l2.e));
    res.x += (l1.e.x - l1.s.x) * t;
    res.y += (l1.e.y - l1.s.y) * t;
    return {2, res};
}
```

5.2.3 Segment-Segment

```
bool segxseg(line l1, line l2)
{
    return
        max(l1.s.x, l1.e.x) >= min(l2.s.x, l2.e.x) &&
        max(l2.s.x, l2.e.x) >= min(l1.s.x, l1.e.x) &&
        max(l1.s.y, l1.e.y) >= min(l2.s.y, l2.e.y) &&
        max(l2.s.y, l2.e.y) >= min(l1.s.y, l1.e.y) &&
        sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e-l1.e) ^ (l1.s - l1.e)) <= 0 &&
        sgn((l1.s - l2.e) ^ (l2.s - l2.e)) * sgn((l1.e-l2.e) ^ (l2.s - l2.e)) <= 0;
}
```

5.2.4 Line-Segment

```
//l1是直线,l2是线段
bool segxline(line l1, line l2)
{
    return sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e - l1.e) ^ (l1.s - l1.e)) <=
    0;
}
```

5.2.5 Point-Line

```
double pointtoline(point p, line l)
{
    point res;
    double t = ((p - l.s) * (l.e - l.s)) / ((l.e - l.s) * (l.e - l.s));
    res.x = l.s.x + (l.e.x - l.s.x) * t, res.y = l.s.y + (l.e.y - l.s.y) * t;
    return dist(p, res);
}
```

5.2.6 Point-Segment

```
double pointtosegment(point p, line l)
{
    point res;
    double t = ((p - l.s) * (l.e - l.s)) / ((l.e - l.s) * (l.e - l.s));
    if (t >= 0 && t <= 1)
        res.x = l.s.x + (l.e.x - l.s.x) * t, res.y = l.s.y + (l.e.y - l.s.y) * t;
    else
        res = dist(p, l.s) < dist(p, l.e) ? l.s : l.e;
    return dist(p, res);
}
```

5.2.7 Point on Segment

```
bool PointOnSeg(point p, line l)
{
    return
        sgn((l.s - p) ^ (l.e-p)) == 0 &&
        sgn((p.x - l.s.x) * (p.x - l.e.x)) <= 0 &&
        sgn((p.y - l.s.y) * (p.y - l.e.y)) <= 0;
}
```

5.3 Polygon

5.3.1 Area

```
double area(point p[], int n)
{
    double res = 0;
    for (int i = 0; i < n; i++) res += (p[i] ^ p[(i + 1) % n]) / 2;
    return fabs(res);
}
```

5.3.2 Point in Convex

```
// 点形成一个凸包，而且按逆时针排序(如果是顺时针把里面的<0改为>0)
// 点的编号 : [0,n)
// -1 : 点在凸多边形外
// 0  : 点在凸多边形边界上
// 1  : 点在凸多边形内
int PointInConvex(point a, point p[], int n)
{
    for (int i = 0; i < n; i++)
        if (sgn((p[i] - a) ^ (p[(i + 1) % n] - a)) < 0)
            return -1;
        else if (PointOnSeg(a, line(p[i], p[(i + 1) % n])))
            return 0;
    return 1;
}
```

5.3.3 Point in Polygon

```
// 射线法,poly[]的顶点数要大于等于3,点的编号0~n-1
// -1 : 点在凸多边形外
// 0  : 点在凸多边形边界上
// 1  : 点在凸多边形内
int PointInPoly(point p, point poly[], int n)
{
    int cnt;
    line ray, side;
    cnt = 0;
    ray.s = p;
    ray.e.y = p.y;
    ray.e.x = -100000000000.0; // -INF,注意取值防止越界
    for (int i = 0; i < n; i++)
    {
        side.s = poly[i], side.e = poly[(i + 1) % n];
        if (PointOnSeg(p, side)) return 0;
        //如果平行轴则不考虑
        if (sgn(side.s.y - side.e.y) == 0)
            continue;
        if (PointOnSeg(sid e.s, r ay))
            cnt += (sgn(side.s.y - side.e.y) > 0);
        else if (PointOnSeg(side.e, ray))
            cnt += (sgn(side.e.y - side.s.y) > 0);
        else if (segxseg(ray, side))
            cnt++;
    }
    return cnt % 2 == 1 ? 1 : -1;
}
```

5.3.4 Judge Convex

```
//点可以是顺时针给出也可以是逆时针给出
//点的编号1~n-1
bool isconvex(point poly[], int n)
{
    bool s[3];
    memset(s, 0, sizeof(s));
    for (int i = 0; i < n; i++)
    {
        s[sgn((poly[(i + 1) % n] - poly[i]) ^ (poly[(i + 2) % n] - poly[i])) + 1] = 1;
        if (s[0] && s[2]) return 0;
    }
    return 1;
}
```

5.4 Integer Points

5.4.1 On Segment

```
int OnSegment(line l) { return __gcd(fabs(l.s.x - l.e.x), fabs(l.s.y - l.e.y)) + 1; }
```

5.4.2 On Polygon Edge

```
int OnEdge(point p[], int n)
{
    int i, ret = 0;
    for (i = 0; i < n; i++)
        ret += __gcd(fabs(p[i].x - p[(i + 1) % n].x), fabs(p[i].y - p[(i + 1) % n].y));
    return ret;
}
```

5.4.3 Inside Polygon

```
int InSide(point p[], int n)
{
    int i, area = 0;
    for (i = 0; i < n; i++)
        area += p[(i + 1) % n].y * (p[i].x - p[(i + 2) % n].x);
    return (fabs(area) - OnEdge(p, n)) / 2 + 1;
}
```

5.5 Circle

5.5.1 Circumcenter

```
point waixin(point a, point b, point c)
{
    double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1 * a1 + b1 * b1) / 2;
    double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2 * a2 + b2 * b2) / 2;
    double d = a1 * b2 - a2 * b1;
    return point(a.x + (c1 * b2 - c2 * b1) / d, a.y + (a1 * c2 - a2 * c1) / d);
}
```

5.6 RuJia Liu's

5.6.1 Point

```
struct Point
{
    double x, y;
    Point(double x = 0, double y = 0) : x(x), y(y) {}
};

typedef Point Vector;

//向量+向量=向量，点+向量=点
Vector operator+(Vector A, Vector B) { return Vector(A.x + B.x, A.y + B.y); }
//点-点=向量
Vector operator-(Point A, Point B) { return Vector(A.x - B.x, A.y - B.y); }
//向量*数=向量
Vector operator*(Vector A, double p) { return Vector(A.x * p, A.y * p); }
//向量/数=向量
Vector operator/(Vector A, double p) { return Vector(A.x / p, A.y / p); }

bool operator<(const Point& a, const Point& b)
{
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}

const double eps = 1e-10;
double dcmp(double x)
{
    if (fabs(x) < eps)
        return 0;
    else
        return x < 0 ? -1 : 1;
}

bool operator==(const Point& a, const Point& b)
{
    return dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y) == 0;
}

/*
 * 基本运算:
 * 点积
 * 叉积
 * 向量旋转
 */
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); }

//rad是弧度
Vector Rotate(Vector A, double rad)
{
    return Vector(A.x * cos(rad) - A.y * sin(rad),
                  A.x * sin(rad) + A.y * cos(rad));
}
```

```

//调用前请确保A不是零向量
Vector Normal(Vector A)
{
    double L = Length(A);
    return Vector(-A.y / L, A.x / L);
}

/*
 * 点和直线:
 * 两直线交点
 * 点到直线的距离
 * 点到线段的距离
 * 点在直线上的投影
 * 线段相交判定
 * 点在线段上判定
 */

//调用前保证两条直线P+tv和Q+tw有唯一交点。当且仅当Cross(v, w)非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;
}

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

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 (dcmp(Dot(v1, v2)) < 0) return Length(v2);
    if (dcmp(Dot(v1, v3)) > 0) return Length(v3);
    return fabs(Cross(v1, v2)) / Length(v1);
}

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

bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2)
{
    double c1 = Cross(a2 - a1, b1 - a1), c2 = Cross(a2 - a1, b2 - b1),
           c3 = Cross(b2 - b1, a1 - b1), c4 = Cross(b2 - b1, a2 - b1);
    return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0;
}

bool OnSegment(Point p, Point a1, Point a2)
{
    return dcmp(Cross(a1 - p, a2 - p)) == 0 && dcmp(Dot(a1 - p, a2 - p)) < 0;
}
```

5.6.2 Circle

```
struct Line
{
    Point p;    //直线上任意一点
    Vector v;   //方向向量。它的左边就是对应的半平面
    double ang; //极角。即从x正半轴旋转到向量v所需要的角（弧度）
    Line() {}
    Line(Point p, Vector v) : p(p), v(v) { ang = atan2(v.y, v.x); }
    bool operator<(const Line& L) const // 排序用的比较运算符
    {
        return ang < L.ang;
    }
    Point point(double t) { return p + v * t; }
};

struct Circle
{
    Point c;
    double r;
    Circle(Point c, double r) : c(c), r(r) {}
    Point point(double a) { return c.x + cos(a) * r, c.y + sin(a) * r; }
};

int getLineCircleIntersection(Line L, Circle C, double& t1, double& t2, vector<Point>& sol)
{
    double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;
    double e = a * a + c * c, f = 2 * (a * b + c * d), g = b * b + d * d - C.r * C.r;
    double delta = f * f - 4 * e * g; //判别式
    if (dcmp(delta) < 0) return 0;    //相离
    if (dcmp(delta) == 0)             //相切
    {
        t1 = t2 = -f / (2 * e);
        sol.push_back(L.point(t1));
        return 1;
    }
    //相交
    t1 = (-f - sqrt(delta)) / (2 * e);
    t2 = (-f + sqrt(delta)) / (2 * e);
    sol.push_back(t1);
    sol.push_back(t2);
    return 2;
}

double angle(Vector v) { return atan2(v.y, v.x); }

int getCircleCircleIntersection(Circle C1, Circle C2, vector<Point>& sol)
{
    double d = Length(C1.c - C2.c);
    if (dcmp(d) == 0)
    {
        if (dcmp(C1.r - C2.r) == 0) return -1; //两圆重合
        return 0;
    }
    if (dcmp(C1.r + C2.r - d) < 0) return 0;        //内含
    if (dcmp(fabs(C1.r - C2.r) - d) > 0) return 0; //外离

    double a = angle(C2.c - C1.c); //向量C1C2的极角
```

```
    double da = acos((C1.r * C1.r + d * d - C2.r * C2.r) / (2 * C1.r * d));
    //C1C2到C1P1的角
    Point p1 = C1.point(a - da), p2 = C1.point(a + da);

    sol.push_back(p1);
    if (p1 == p2) return 1;
    sol.push_back(p2);
    return 2;
}

//过点p到圆C的切线，v[i]是第i条切线的向量，返回切线条数
int getTangents(Point p, Circle C, Vector* v)
{
    Vector u = C.c - p;
    double dist = Length(u);
    if (dist < C.r)
        return 0;
    else if (dcmp(dist - C.r) == 0)
    { //p在圆上，只有一条切线
        v[0] = Rotate(u, M_PI / 2);
        return 1;
    }
    else
    {
        double ang = asin(C.r / dist);
        v[0] = Rotate(u, -ang);
        v[1] = Rotate(u, +ang);
        return 2;
    }
}

//两圆的公切线
//返回切线的条数。-1表示无穷条切线。
//a[i]和b[i]分别是第i条切线在圆A和圆B上的切点
int getTangents(Circle A, Circle B, Point* a, Point* b)
{
    int cnt = 0;
    if (A.r < B.r)
    {
        swap(A, B);
        swap(a, b);
    }
    int d2 = (A.c.x - B.c.x) * (A.c.x - B.c.x) + (A.c.y - B.c.y) * (A.c.y - B.c.y);
    int rdiff = A.r - B.r;
    int rsum = A.r + B.r;
    if (d2 < rdiff * rdiff) return 0; //内含
    double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
    if (d2 == 0 && A.r == B.r) return -1; //无限多条切线
    if (d2 == rdiff * rdiff)
    { //内切，一条切线
        a[cnt] = A.point(base);
        b[cnt] = B.point(base);
        cnt++;
        return 1;
    }
    //有外共切线
    double ang = acos(A.r - B.r) / sqrt(d2);
    a[cnt] = A.point(base + ang);
    b[cnt] = B.point(base + ang);
```

```
    cnt++;
    a[cnt] = A.point(base + ang);
    b[cnt] = B.point(base - ang);
    cnt++;
    if (d2 == rsum * rsum)
    {
        a[cnt] = A.point(base);
        b[cnt] = B.point(M_PI + base);
        cnt++;
    }
    else if (d2 > rsum * rsum)
    {
        double ang = acos((A.r + B.r) / sqrt(d2));
        a[cnt] = A.point(base + ang);
        b[cnt] = B.point(M_PI + base + ang);
        cnt++;
        a[cnt] = A.point(base - ang);
        b[cnt] = B.point(M_PI + base - ang);
        cnt++;
    }
    return cnt;
}

//三角形外接圆（三点保证不共线）
Circle CircumscribedCircle(Point p1, Point p2, Point p3)
{
    double Bx = p2.x - p1.x, By = p2.y - p1.y;
    double Cx = p3.x - p1.x, Cy = p3.y - p1.y;
    double D = 2 * (Bx * Cy - By * Cx);
    double cx = (Cy * (Bx * Bx + By * By) - By * (Cx * Cx + Cy * Cy)) / D + p1.x;
    double cy = (Bx * (Cx * Cx + Cy * Cy) - Cx * (Bx * Bx + By * By)) / D + p1.y;
    Point p = Point(cx, cy);
    return Circle(p, Length(p1 - p));
}

//三角形内切圆
Circle InscribedCircle(Point p1, Point p2, Point p3)
{
    double a = Length(p2 - p3);
    double b = Length(p3 - p1);
    double c = Length(p1 - p2);
    Point p = (p1 * a + p2 * b + p3 * c) / (a + b + c);
    return Circle(p, DistanceToLine(p, p1, p2));
}
```

5.6.3 Polygon

```
typedef vector<Point> Polygon;
//多边形的有向面积
double PolygonArea(Polygon po)
{
    int n = po.size();
    double area = 0.0;
    for (int i = 1; i < n - 1; i++)
        area += Cross(po[i] - po[0], po[i + 1] - po[0]);
    return area / 2;
}

```

```
//点在多边形内判定
int isPointInPolygon(Point p, Polygon poly)
{
    int wn = 0; //绕数
    int n = poly.size();
    for (int i = 0; i < n; i++)
    {
        if (OnSegment(p, poly[i], poly[(i + 1) % n])) return -1; //边界上
        int k = dcmp(Cross(poly[(i + 1) % n] - poly[i], p - poly[i]));
        int d1 = dcmp(poly[i].y - p.y);
        int d2 = dcmp(poly[(i + 1) % n].y - p.y);
        if (k > 0 && d1 <= 0 && d2 > 0) wn++;
        if (k < 0 && d2 <= 0 && d1 > 0) wn--;
    }
    if (wn != 0) return 1; //内部
    return 0;              //外部
}

//凸包(Andrew算法)
//如果不希望在凸包的边上有输入点，把两个 <= 改成 <
//如果不介意点集被修改，可以改成传递引用
Polygon ConvexHull(vector<Point> p)
{
    sort(p.begin(), p.end());
    p.erase(unique(p.begin(), p.end()), p.end());
    int n = p.size(), m = 0;
    Polygon res(n + 1);
    for (int i = 0; i < n; i++)
    {
        while (m > 1 && Cross(res[m - 1] - res[m - 2], p[i] - res[m - 2]) <= 0) m--;
        res[m++] = p[i];
    }
    int k = m;
    for (int i = n - 2; i >= 0; i--)
    {
        while (m > k && Cross(res[m - 1] - res[m - 2], p[i] - res[m - 2]) <= 0) m--;
        res[m++] = p[i];
    }
    m -= n > 1;
    res.resize(m);
    return res;
}

//半平面交
vector<Point> HalfplaneIntersection(vector<Line>& L)
{
    int n = L.size();
    sort(L.begin(), L.end()); // 按极角排序

    int first, last;     // 双端队列的第一个元素和最后一个元素的下标
    vector<Point> p(n); // p[i]为q[i]和q[i+1]的交点
    vector<Line> q(n);  // 双端队列
    vector<Point> ans;  // 结果

    q[first = last = 0] = L[0]; // 双端队列初始化为只有一个半平面L[0]
    for (int i = 1; i < n; i++)
    {
        while (first < last && !OnLeft(L[i], p[last - 1])) last--;
        while (first < last && !OnLeft(L[i], p[first])) first++;
```

```
        q[++last] = L[i];
        if (fabs(Cross(q[last].v, q[last - 1].v)) < eps)
        { // 两向量平行且同向，取内侧的一个
            last--;
            if (OnLeft(q[last], L[i].p)) q[last] = L[i];
        }
        if (first < last) p[last - 1] = GetLineIntersection(q[last - 1], q[last]);
    }
    while (first < last && !OnLeft(q[first], p[last - 1])) last--; // 删除无用平面
    if (last - first <= 1) return vector<Point>();                  // 空集
    p[last] = GetLineIntersection(q[last], q[first]);               // 计算首尾两个半平面的交点

    return vector<Point>(q.begin() + first, q.begin() + last + 1);
}
```

6 Dynamic Programming

6.1 Subsequence

6.1.1 Max Sum

```
// 传入序列a和长度n，返回最大子序列和
int MaxSeqSum(int a[], int n)
{
    int rt = 0, cur = 0;
    for (int i = 0; i < n; i++)
        cur += a[i], rt = max(cur, rt), cur = max(0, cur);
    return rt;
}
```

6.1.2 Longest Increase

```
// 序列下标从1开始，LIS()返回长度，序列存在lis[]中
const int N = "Edit";
int len, a[N], b[N], f[N];
int Find(int p, int l, int r)
{
    while (l <= r)
    {
        int mid = (l + r) >> 1;
        if (a[p] > b[mid])
            l = mid + 1;
        else
            r = mid - 1;
    }
    return f[p] = l;
}
int LIS(int lis[], int n)
{
    int len = 1;
    f[1] = 1, b[1] = a[1];
    for (int i = 2; i <= n; i++)
    {
        if (a[i] > b[len])
            b[++len] = a[i], f[i] = len;
        else
            b[Find(i, 1, len)] = a[i];
    }
    for (int i = n, t = len; i >= 1 && t >= 1; i--)
        if (f[i] == t) lis[--t] = a[i];
    return len;
}

// 简单写法(下标从0开始,只返回长度)
int dp[N];
int LIS(int a[], int n)
{
    memset(dp, 0x3f, sizeof(dp));
    for (int i = 0; i < n; i++) *lower_bound(dp, dp + n, a[i]) = a[i];
    return lower_bound(dp, dp + n, INF) - dp;
}
```

6.1.3 Longest Common Increase

```cpp
// 序列下标从1开始
int LCIS(int a[], int b[], int n, int m)
{
    memset(dp, 0, sizeof(dp));
    for (int i = 1; i <= n; i++)
    {
        int ma = 0;
        for (int j = 1; j <= m; j++)
        {
            dp[i][j] = dp[i - 1][j];
            if (a[i] > b[j]) ma = max(ma, dp[i - 1][j]);
            if (a[i] == b[j]) dp[i][j] = ma + 1;
        }
    }
    return *max_element(dp[n] + 1, dp[n] + 1 + m);
}
```

6.2 Digit Statistics

```cpp
int a[20];
ll dp[20][state];
ll dfs(int pos, /*state变量*/, bool lead /*前导零*/, bool limit /*数位上界变量*/)
{
    //递归边界，既然是按位枚举，最低位是0，那么pos==-1说明这个数枚举完了
    if (pos == -1) return 1;
    /*这里一般返回1，表示枚举的这个数是合法的，那么这里就需要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。*/
    if (!limit && !lead && dp[pos][state] != -1) return dp[pos][state];
    /*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*/
    int up = limit ? a[pos] : 9; //根据limit判断枚举的上界up
    ll ans = 0;
    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
    {
        if () ...
        else if () ...
        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
        /*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
        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*/
    }
    //计算完，记录状态
    if (!limit && !lead) dp[pos][state] = ans;
    /*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*/
    return ans;
}
ll solve(ll x)
{
    int pos = 0;
    do //把数位都分解出来
        a[pos++] = x % 10;
    while (x /= 10);
    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
}
```

6.3 Slope Optimization

问题 设 $f(i) = \min(y[k] - s[i] \times x[k]), k \in [1, i-1]$, 现在要求出所有 $f(i), i \in [1, n]$

考虑两个决策 j 和 k，如果 j 比 k 优，则

$$y[j] - s[i] \times x[j] < y[k] - s[i] \times x[k]$$

化简得:

$$\frac{y_j - y_k}{x_j - x_k} < s_i$$

不等式左边是个斜率，我们把它设为 $\mathrm{slope}(j, k)$

我们可以维护一个单调递增的队列，为什么呢?

因为如果 $\mathrm{slope}(q[i-1], q[i]) > \mathrm{slope}(q[i], q[i+1])$，那么当前者成立时，后者必定成立。即 $q[i]$ 决策优于 $q[i-1]$ 决策时，$q[i+1]$ 必然优于 $q[i]$，因此 $q[i]$ 就没有存在的必要了。所以我们要维护递增的队列。

那么每次的决策点 i，都要满足

$$\begin{cases} \mathrm{slope}(q[i-1], q[i]) < s[i] \\ \mathrm{slope}(q[i], q[i+1]) \geq s[i] \end{cases}$$

一般情况去二分这个 i 即可。

如果 $s[i]$ 是单调不降的，那么对于决策 j 和 $k(j < k)$ 来说，如果决策 k 优于决策 j，那么对于 $i \in [k+1, n]$，都存在决策 k 优于决策 j，因此决策 j 就可以舍弃了。这样的话我们可以用单调队列进行优化，可以少个 log。

单调队列滑动窗口最大值

```cpp
// k为滑动窗口的大小
deque<int> q;
for (int i = 0, j = 0; i + k <= d; i++)
{
    while (j < i + k)
    {
        while (!q.empty() && a[q.back()] < a[j]) q.pop_back();
        q.push_back(j++);
    }
    while (q.front() < i) q.pop_front();
    // a[q.front()]为当前滑动窗口的最大值
}
```

7 Others

7.1 Matrix

7.1.1 Matrix FastPow

```
typedef vector<ll> vec;
typedef vector<vec> mat;
mat mul(mat& A, mat& B)
{
    mat C(A.size(), vec(B[0].size()));
    for (int i = 0; i < A.size(); i++)
        for (int k = 0; k < B.size(); k++)
            if (A[i][k]) // 对稀疏矩阵的优化
                for (int j = 0; j < B[0].size(); j++)
                    C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % mod;
    return C;
}
mat Pow(mat A, ll n)
{
    mat B(A.size(), vec(A.size()));
    for (int i = 0; i < A.size(); i++) B[i][i] = 1;
    for (; n; n >>= 1, A = mul(A, A))
        if (n & 1) B = mul(B, A);
    return B;
}
```

7.1.2 Gauss Elimination

```
void gauss()
{
    int now = 1, to;
    double t;
    for (int i = 1; i <= n; i++, now++)
    {
        /*for (to = now; !a[to][i] && to <= n; to++);
        //做除法时减小误差，可不写
        if (to != now)
            for (int j = 1; j <= n + 1; j++)
                swap(a[to][j], a[now][j]);*/
        t = a[now][i];
        for (int j = 1; j <= n + 1; j++) a[now][j] /= t;
        for (int j = 1; j <= n; j++)
            if (j != now)
            {
                t = a[j][i];
                for (int k = 1; k <= n + 1; k++) a[j][k] -= t * a[now][k];
            }
    }
}
```

7.2 Tricks

7.2.1 Stack-Overflow

```
// 解决爆栈问题
#pragma comment(linker, "/STACK:1024000000,1024000000")
```

7.2.2 Fast-Scanner

```
// 适用于正负整数
template <class T>
inline bool scan_d(T &ret)
{
    char c;
    int sgn;
    if (c = getchar(), c == EOF) return 0; //EOF
    while (c != '-' && (c < '0' || c > '9')) c = getchar();
    sgn = (c == '-') ? -1 : 1;
    ret = (c == '-') ? 0 : (c - '0');
    while (c = getchar(), c >= '0' && c <= '9') ret = ret * 10 + (c - '0');
    ret *= sgn;
    return 1;
}
inline void out(int x)
{
    if (x > 9) out(x / 10);
    putchar(x % 10 + '0');
}
```

7.2.3 Strok-Sscanf

```
// 空格作为分隔输入,读取一行的整数
fgets(buf, BUFSIZE, stdin);
int v;
char *p = strtok(buf, " ");
while (p)
{
    sscanf(p, "%d", &v);
    p = strtok(NULL," ");
}
```

7.3 Mo Algorithm

莫队算法, 可以解决一类静态, 离线区间查询问题。分成 $\sqrt{x}$ 块, 分块排序。

```
struct query { int L, R, id; };
void solve(query node[], int m)
{
    memset(ans, 0, sizeof(ans));
    sort(node, node + m, [](query a, query b) {
        return a.l / unit < b.l / unit
               || a.l / unit == b.l / unit && a.r < b.r;
    });
    int L = 1, R = 0;
    for (int i = 0; i < m; i++)
    {
        while (node[i].L < L) add(a[--L]);
        while (node[i].L > L) del(a[L++]);
        while (node[i].R < R) del(a[R--]);
        while (node[i].R > R) add(a[++R]);
        ans[node[i].id] = tmp;
    }
}
```

7.4 BigNum

7.4.1 High-precision

```
// 加法 乘法 小于号 输出
struct bint
{
    int l;
    short int w[100];
    bint(int x = 0)
    {
        l = x == 0, memset(w, 0);
        while (x) w[l++] = x % 10, x /= 10;
    }
    bool operator<(const bint& x) const
    {
        if (l != x.l) return l < x.l;
        int i = l - 1;
        while (i >= 0 && w[i] == x.w[i]) i--;
        return (i >= 0 && w[i] < x.w[i]);
    }
    bint operator+(const bint& x) const
    {
        bint ans;
        ans.l = l > x.l ? l : x.l;
        for (int i = 0; i < ans.l; i++)
        {
            ans.w[i] += w[i] + x.w[i];
            ans.w[i + 1] += ans.w[i] / 10;
            ans.w[i] = ans.w[i] % 10;
        }
        if (ans.w[ans.l] != 0) ans.l++;
        return ans;
    }
    bint operator*(const bint& x) const
    {
        bint res;
        int up, tmp;
        for (int i = 0; i < l; i++)
        {
            up = 0;
            for (int j = 0; j < x.l; j++)
            {
                tmp = w[i] * x.w[j] + res.w[i + j] + up;
                res.w[i + j] = tmp % 10;
                up = tmp / 10;
            }
            if (up != 0) res.w[i + x.l] = up;
        }
        res.l = l + x.l;
        while (res.w[res.l - 1] == 0 && res.l > 1) res.l--;
        return res;
    }
    void print()
    {
        for (int i = l - 1; ~i; i--) printf("%d", w[i]);
        puts("");
    }
};
```

7.4.2 Complete High-precision

```java
import java.math.BigInteger;
```

7.5 Misc

7.5.1 Standard Template Library

```cpp
template <class InputIterator, class OutputIterator>
  OutputIterator copy (InputIterator first, InputIterator last, OutputIterator result);

template <class InputIterator1, class InputIterator2,
          class OutputIterator, class Compare>
  OutputIterator merge (InputIterator1 first1, InputIterator1 last1,
                        InputIterator2 first2, InputIterator2 last2,
                        OutputIterator result, Compare comp);

template <class InputIterator, class Function>
   Function for_each (InputIterator first, InputIterator last, Function fn);

template <class InputIterator, class OutputIterator, class UnaryOperation>
  OutputIterator transform (InputIterator first1, InputIterator last1,
                            OutputIterator result, UnaryOperation op);

template< class ForwardIterator, class T >
void iota( ForwardIterator first, ForwardIterator last, T value );
```

7.5.2 Policy-Based Data Structures

红黑树

声明/头文件

```cpp
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
typedef tree<pt, null_type, less<pt>, rb_tree_tag, tree_order_statistics_node_update>
    rbtree;
```

使用方法

```cpp
pt                                  // 关键字类型
null_type                           // 无映射(低版本g++为null_mapped_type)
less<int>                           // 从小到大排序
rb_tree_tag                         // 红黑树（splay_tree_tag）
tree_order_statistics_node_update   // 结点更新
T.insert(val);                      // 插入
T.erase(iterator);                  // 删除
T.order_of_key();                   // 查找有多少数比它小
T.find_by_order(k);                 // 有k个数比它小的数是多少
a.join(b);                          // b并入a 前提是两棵树的key的取值范围不相交
a.split(v, b);                      // key小于等于v的元素属于a，其余的属于b
T.lower_bound(x);                   // >=x的min的迭代器
T.upper_bound((x);                  // >x的min的迭代器
```

7.5.3 Subset Enumeration

枚举真子集

```
for (int s = (S - 1) & S; s; s = (s - 1) & S)
```

枚举大小为 k 的子集

```
void subset(int k, int n)
{
    int t = (1 << k) - 1;
    while (t < (1 << n))
    {
        // do something
        int x = t & -t, y = t + x;
        t = ((t & ~y) / x >> 1) | y;
    }
}
```

7.5.4 Date Magic

```
string dayOfWeek[] = {"Mo", "Tu", "We", "Th", "Fr", "Sa", "Su"};

// converts Gregorian date to integer (Julian day number)
int DateToInt(int m, int d, int y)
{
    return 1461 * (y + 4800 + (m - 14) / 12) / 4
           + 367 * (m - 2 - (m - 14) / 12 * 12) / 12
           - 3 * ((y + 4900 + (m - 14) / 12) / 100) / 4
           + d - 32075;
}

// converts integer (Julian day number) to Gregorian date: month/day/year
void IntToDate(int jd, int& m, int& d, int& y)
{
    int x, n, i, j;
    x = jd + 68569;
    n = 4 * x / 146097;
    x -= (146097 * n + 3) / 4;
    i = (4000 * (x + 1)) / 1461001;
    x -= 1461 * i / 4 - 31;
    j = 80 * x / 2447;
    d = x - 2447 * j / 80;
    x = j / 11;
    m = j + 2 - 12 * x;
    y = 100 * (n - 49) + i + x;
}

// converts integer (Julian day number) to day of week
string IntToDay(int jd) { return dayOfWeek[jd % 7]; }
```

7.6 Configuration

7.6.1 VSCode

launch.json

```
{
    "version": "0.2.0",
    "configurations": [
```

```
        {
            "name": "(gdb) Launch",
            "type": "cppdbg",
            "request": "launch",
            "program": "${workspaceRoot}/a.out",
            "args": [],
            "stopAtEntry": false,
            "cwd": "${fileDirname}",
            "environment": [],
            "externalConsole": true,
            "MIMode": "gdb",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                }
            ],
            "preLaunchTask": "build"
        }
    ]
}
```

task.json

```
{
    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "2.0.0",
    "tasks": [
        {
            "label": "build",
            "type": "shell",
            "command": "g++",
            "args": [
                "-g",
                "-std=c++17",
                "${file}"
            ],
            "group": {
                "kind": "build",
                "isDefault": true
            },
            "problemMatcher": {
                "owner": "cpp",
                "fileLocation": "absolute",
                "pattern": {
                    "regexp": "^(.*):(\\d+):(\\d+):\\s+(warning|error):\\s+(.*)$",
                    "file": 1,
                    "line": 2,
                    "column": 3,
                    "severity": 4,
                    "message": 5
                }
            }
        }
    ]
}
```

7.6.2 Vim

```
syntax on
set cindent
set nu
set tabstop=4
set shiftwidth=4
set background=dark
set mouse=a

map<C-A> ggvG"+y
map<F5> :call Run()<CR>

func! Run()
    exec "w"
    exec "!g++ -std=c++11 -O2 % -o %<"
    exec "!time ./%<"
endfunc

autocmd BufNewFile *.cpp 0r ~/include.cpp
autocmd BufNewFile *.cpp normal G

inoremap ( ()<Esc>i
inoremap [ []<Esc>i
inoremap { {<CR>}<Esc>O
inoremap ' ''<Esc>i
inoremap " ""<Esc>i

inoremap ) <c-r>=ClosePair(')')<CR>
inoremap ] <c-r>=ClosePair(']')<CR>

func ClosePair(char)
    if getline('.')[col('.')-1]==a:char
        return "\<Right>"
    else
        return a:char
    endif
endfunc
```