| cpggの模版集

这道题可以打表吗?

有必要关longlong吗?

IOS开了吗?

数据范围真的开够了吗?

测测最小样例和最大样例

头部

```
//#include<bits/stdc++.h>
#include<iostream> <iomanip> <vector> <cmath> <cstring> <stack> <map> <queue>
<algorithm> <functional> <bitset> <random>
#define PQ priority_queue
typedef long long i64;

//#pragma GCC optimize(1)
//#pragma GCC optimize(2)
//#pragma GCC optimize(3,"Ofast","inline")

#define IOS std::ios_base::sync_with_stdio(false), std::cin.tie(0);
int IOS = [] {
    std::ios_base::sync_with_stdio(0), std::cin.tie(0);
    return 0;
}();
//dev : 工具 -> 编译选项 -> "-std=c++14" -> 打勾 -> 应用 -> ctrl+n
//小心div是关键字
#define count(x) __builtin_popcount(x)
// Rope
#include <bits/extc++.h>
using namespace __gnu_cxx;
#define pb __gnu_pbds // 这个是pbds的
// 这些都要求g++编译器, clang++不支持
```

[TOC]

1, 数学

1-1 大质数表

image-20230119124801993

image-20230119124818295

image-20230119124832156

image-20230119124844916

image-20230712103538296

1e9+21, 1e9+33 , 1e9+87, 1e9+93, 1e9+97, 质数！

$w(n)$ 指质因数数量, $d(n)$ 指约数数量

常数 e = 2.71828182845904523536028747135266249775724

```
// 18位素数：154590409516822759
// 19位素数：2305843009213693951 (梅森素数)
// 19位素数：4384957924686954497
```

1-2 米勒罗宾判素数

```
/*
维基百科 :
n < 4e9, prime = [2, 7, 61]
n < 3e14, prime = [2, 3, 5, 7, 11, 13, 17]
n < 3e18, prime = [2, 3, 5, 7, 11, 13, 17, 19, 23]
n < 3e23, prime = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
*/
constexpr int prime[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
// 快速乘
i64 mulp(i64 x, i64 y, const i64 &mod) {
    return (__int128)x * y % mod; // 还是尽量用__int128, 龟速乘实在是太慢了
    // i64 ans = 0;
    // for (; y; y >>= 1) {
    //     if (y & 1)
    //         ans = (ans + x) % mod;
    //     x = (x + x) % mod;
    // }
    // return ans;
}
i64 powp(i64 a, i64 mi, const i64 &mod) {
    i64 ans = 1;
    for (; mi; mi >>= 1) {
        if (mi & 1)
            ans = mulp(ans, a, mod);
        a = mulp(a, a, mod);
    }
    return ans;
}

bool isp(i64 v) { // 判断v是不是质数
    if (v < 2 or v != 2 and v % 2 == 0)
        return false;
    i64 s = v - 1;
    while (!(s & 1))
        s >>= 1;
    for (int x : prime) {
        if (v == x)
            return true;
        i64 t = s, m = powp(x, s, v);
        while (t != v - 1 and m != 1 and m != v - 1)
            m = mulp(m, m, v), t <<= 1;
        if (m != v - 1 and !(t & 1))
            return false;
    }
    return true;
}
```

1-3 取随机数

```
std::mt19937 myrand(time(0));
int rd(int l, int r){
    return std::uniform_int_distribution<int>(l, r)(myrand);
}
//得到[l,r]之间的均匀随机数
```

1-4 Pollard rho 算法

如果 n 是质数 $(Miller-Rabbin判)$ 返回 n

否则返回 n 的随机一个 $[2,n-1]$ 的因子

复杂度理论 $O(n^{\frac{1}{4}}\log n)$ 但实际跑得快, 可以按 $O(n^{\frac{1}{4}})$ 算

```
// 使用前提: 把上面的米勒罗宾抄下来
i64 gcd(i64 a, i64 b) {
    return b == 0 ? a : gcd(b, a % b);
}
std::mt19937 myrand(time(0));
i64 rd(i64 l, i64 r) { // 注意这里是i64的随机数
    return std::uniform_int_distribution<i64>(l, r)(myrand);
}
i64 Pollard_Rho(i64 n) { // 返回 n 的随机一个[2, n-1]内的因子, 或者判定是质数
    if (n == 4)
        return 2;
    if (isp(n)) // Miller-Rabbin 判质数
        return n; // 如果 n 是质数直接返回 n
    while (true) {
        i64 c = rd(1, n - 1);
        auto f = [=](i64 x) { return (mulp(x, x, n) + c) % n; };
        i64 t = 0, r = 0, p = 1, q;
        do {
            for (int i = 0; i < 128; i++) {
                t = f(t), r = f(f(r));
                if (t == r || (q = mulp(p, std::abs(t - r), n)) == 0)
                    break;
                p = q;
            }
            i64 d = gcd(p, n);
            if (d > 1)
                return d;
        } while (t != r);
    }
}
```

1-5 求所有约数

```
std::vector<int> div; // div是关键字
void getdiv(int x) {
    div.clear();
    if (x == 0)
        return;
    div.push_back(1);
    while (x > 1) {
        int pr = v[x];
        int l = 0, r = div.size();
        while (x % pr == 0) {
            x /= pr;
            for (int k = l; k < r; k++)
                div.push_back(div[k] * pr);
            l = r, r = div.size();
        }
    }
}
```

1-6 $\varphi(x)$ 和 $\mu (x)$

```
int phi[N], mu[N];
void euler(){
    std::iota(phi, phi + N, 0), mu[1] = 1;
    for (int i = 1; i < N; i++)
        for (int j = i + i; j < N; j += i)
            phi[j] -= phi[i], mu[j] -= mu[i];
}
```

1-7 指数方程

```
int exeq(int a, int g) {
    // a ^ n = g (mod p) 求最小的n
    int sq = sqrt(p) + 1;
    std::unordered_map<int, int> mp;
    int b = 1, ans = 1 << 30;
    for (int i = 0; i < sq; i++) {
        if (!mp.count(b))
            mp[b] = i;
        b = 1ll * a * b % p;
    }
    int invb = powp(b, p - 2), tar = g;
    for (int i = 0; i * sq <= p; i++) {
        if (mp.count(tar))
            ans = std::min(ans, mp[tar] + i * sq);
        tar = 1ll * tar * invb % p;
    }
    if (ans == (1 << 30))
        ans = -1; // 无解
    return ans;
}
```

1-8 线性逆元

```
inv[1] = 1;
for (int i = 2; i <= n; i++)
    inv[i] = 1ll * (p - p / i) * inv[p % i] % p;
```

1-9 排列组合

```
int powp(int a, int mi) {
    int ans = 1;
    for (; mi; mi >>= 1, a = 1ll * a * a % p)
        if (mi & 1)
            ans = 1ll * ans * a % p;
    return ans;
}
int jc[N], ijc[N], Capps = [] {
    jc[0] = ijc[0] = 1;
    for (int i = 1; i < N; i++)
```

```
        jc[i] = 1ll * jc[i - 1] * i % p;
    ijc[N - 1] = powp(jc[N - 1], p - 2);
    for (int i = N - 1; i; i--)
        ijc[i - 1] = 1ll * ijc[i] * i % p;
    return 0;
}(); // O(n) 预处理
int A(int a, int b) {
    return (a < b ? 0 : 1ll * jc[a] * ijc[a - b] % p);
}
int C(int a, int b) {
    return (a < b ? 0 : 1ll * jc[a] * ijc[a - b] % p * ijc[b] % p);
}// O(1) 查询
```

1-10 原根

image-20230510194225386 image-20230420125137419

其中$\phi(x)$是欧拉函数, 以下代码可以一次性求 10^6 以下的数的原根

```
int getGenRoot(int x) {
    int phi = (x % 2 == 0 ? x / 2 : x - 1);
    for (int g = 2;; g++) {
        int ok = 1, h = g, i = 1;
        for (; h != 1; i++)
            h = 1ll * h * g % x;
        if (i == phi)
            return g;
    }
}
```

image-20230807210408338

10^9+7 的原根是 5

1-11 NTT

```
#define VI std::vector<int>
//唯一需要调的参数是LG

const int LG = 21; //内存参数，约为log(多项式长度)      LG==21 -> 30MB (注意调节)
const int r = (1 << LG);
const int p = 998244353; // p+p不能溢出
inline int mul(const int &a, const int &b) {
    return 1ll * a * b % p; // int
}
int powp(int a, int b = p - 2) {
    int c = 1;
    for (; b; b /= 2, a = mul(a, a))
        if (b & 1)
            c = mul(c, a);
    return c;
}
int w[r], ny[r], Capps = [] {
    w[r / 2] = 1;
    int s = powp(3, (p - 1) / r);
    for (int i = r / 2 + 1; i < r; ++i)
        w[i] = mul(s, w[i - 1]);
    for (int i = r / 2 - 1; i > 0; --i)
        w[i] = w[i * 2];
    ny[1] = 1;
    for (int i = 2; i < r; i++)
        ny[i] = 1ll * (p - p / i) * ny[p % i] % p;
    return 0;
}();
void dft(int *a, int n) {
    for (int k = n / 2; k; k /= 2)
        for (int i = 0; i < n; i += 2 * k)
            for (int j = 0; j < k; ++j) {
                int t = a[i + j + k];
                a[i + j + k] = mul(a[i + j] - t + p, w[k + j]);
                a[i + j] = (a[i + j] + t) % p;
            }
}
void idft(int *a, int n) {
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k)
            for (int j = 0; j < k; ++j) {
                int u = a[i + j];
                int v = mul(a[i + j + k], w[j + k]);
                a[i + j + k] = (u + p - v) % p;
                a[i + j] = (u + v) % p;
            }
    for (int i = 0; i < n; ++i)
        a[i] = mul(a[i], p - (p - 1) / n);
    std::reverse(a + 1, a + n);
}
VI operator*(VI a, VI b) {
    int n = a.size() + b.size() - 1;
    int s = 1 << std::__lg(2 * n - 1);
    a.resize(s);
    b.resize(s);
    dft(a.data(), a.size());
    dft(b.data(), b.size());
    for (int i = 0; i < s; ++i)
        a[i] = mul(a[i], b[i]);
    idft(a.data(), a.size());
    return a.resize(n), a;
}
//用法：引入两个VI f和g, 直接乘f*g返回二者乘积的VI
//注意vector表示多项式中，第0项表示常数项
//这里的模数只能是998244353或者1004535809
VI inv(const VI &a) {
    // 常数项必须是 1
    int n = a.size();
    if (n == 1) {
        return {powp(a[0], p - 2)};
    }
    VI a2(a.begin(), a.begin() + (n + 1) / 2);
    VI b = inv(a2);
    VI c = a * b;
    for (int &i : c)
        i = (p - i) % p;
    c[0] = (c[0] + 2) % p;
    c = c * b;
    return c.resize(n), c;
}
VI ln(const VI &a) {
    // 常数项必须是 1
    int n = a.size();
    VI b(n, 0);
    for (int i = 1; i < n; i++)
        b[i - 1] = 1ll * i * a[i] % p;
    b = b * inv(a);
    b.resize(n);
    for (int i = n - 1; i; i--)
        b[i] = 1ll * b[i - 1] * ny[i] % p;
    b[0] = 0;
    return b;
}
VI exp(const VI &a) {
    int n = a.size();
    if (n == 1) {
        return {1};
    }
    VI a2(a.begin(), a.begin() + (n + 1) / 2);
    VI b = exp(a2);
    b.resize(n);
    VI c = ln(b);
    for (int i = 0; i < n; i++)
        c[i] = (a[i] - c[i] + p) % p;
    c[0] = (c[0] + 1) % p;
    c = c * b;
    return c.resize(n), c;
}
```

1-12 FFT

```
//常常要开longlong
typedef double ld; // double 就够了,10^15也没必要开longdouble
#define pdd std::pair<ld, ld>
#define vd std::vector<ld>
#define VI std::vector<int>
#define x first
#define y second
//调参
const int EX = 21; // 21 能支持俩1e6多项式的乘法
const ld pi = 3.1415926535897932384626433832795028841971;
// Complex number
pdd operator+(const pdd &a, const pdd &b) {
    return {a.x + b.x, a.y + b.y};
}
pdd operator-(const pdd &a, const pdd &b) {
    return {a.x - b.x, a.y - b.y};
}
pdd operator*(const pdd &a, const pdd &b) {
    return {a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x};
}
int l, limit = 1, r[1 << EX];
pdd Wn[1 << EX];
void FFT(pdd *a, int opt) {
    for (int i = 0; i < limit; i++)
        if (i < r[i])
            swap(a[i], a[r[i]]);
    for (int mid = 1; mid < limit; mid <<= 1) {
        for (int i = 0; i < mid; i++)
            Wn[i] = {cosl(pi * i / mid), opt * sinl(pi * i / mid)};
        for (int R = mid << 1, j = 0; j < limit; j += R) {
            for (int k = 0; k < mid; k++) {
                pdd x = a[j + k], y = Wn[k] * a[j + mid + k];
                a[j + k] = x + y;
                a[j + mid + k] = x - y;
            }
        }
    }
}
/*
```

```
精度较低的写法,记得要把Wn数组删了
void FFT(pdd *a, int opt) {
    for (int i = 0; i < limit; i++)
        if (i < r[i])
            swap(a[i], a[r[i]]);
    for (int mid = 1; mid < limit; mid <<= 1) {
        pdd Wn = {cos(pi / mid), opt * sin(pi / mid)};
        for (int R = mid << 1, j = 0; j < limit; j += R) {
            pdd w = {1, 0};
            for (int k = 0; k < mid; k++, w = w * Wn) {
                pdd x = a[j + k], y = w * a[j + mid + k];
                a[j + k] = x + y;
                a[j + mid + k] = x - y;
            }
        }
    }
}
*/
VI operator*(const VI &a, const VI &b) {
    int n = a.size(), m = b.size();
    //要注意结果int装不装的下
    VI ans(n + m - 1);
    n--, m--;
    limit = 1;
    l = 0;
    while (limit <= n + m)
        limit <<= 1, l++;
    for (int i = 1; i <= limit; i++)
        r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));
    //---可换---------
    pdd *c = new pdd[limit]();
    for (int i = 0; i <= n; i++)
        c[i].x = a[i];
    for (int i = 0; i <= m; i++)
        c[i].y = b[i];
    FFT(c, 1);
    for (int i = 0; i < limit; i++)
        c[i] = c[i] * c[i];
    FFT(c, -1);
    for (int i = 0; i <= n + m; i++)
        ans[i] = c[i].y / (limit << 1) + 0.5;
    delete[] c;
    //---可换---------
    return ans;
}
//如果要求浮点数卷积,可以将"可换"区间换成
/*
    pdd*c=new pdd[limit]();
    pdd*v=new pdd[limit]();
    for(int i=0; i<=n; i++) c[i].x=a[i];
    for(int i=0; i<=m; i++) v[i].x=b[i];
    FFT(c,1); FFT(v,1);
    for(int i=0; i<limit; i++)c[i]=c[i]*v[i];
    FFT(c,-1);
    for(int i=0; i<=n+m; i++) ans[i]=c[i].x/limit;
    delete[]c,v;
    //注意要把VI都换成vd
*/

//用法：引入两个VI f和g, 直接乘f*g返回二者乘积的VI
//注意vector表示多项式中，第0项表示常数项
//注意卷积结果不要太大，太大的话得考虑开longlong
```

1-13分治FFT

```
int f[N], g[N];
/*
 g, f[0] 已知
 f[i] = sum(f[j]*g[k])
      j+k≤i-1
*/
void divide(int l, int r) {
    if (l == r){
        // 如果式子复杂就在这里进行真值转化
        return;
    }
    int mid = l + r >> 1;
    divide(l, mid);
    VI a(mid - l + 1), b(r - l + 1);
    for (int i = l; i <= mid; i++)
        a[i - l] = f[i];
    for (int i = l; i <= r; i++)
        b[i - l] = g[i - l];
    auto c = a * b;
    for (int i = mid + 1; i <= r; i++)
        f[i] = (f[i] + c[i - l - 1]) % p;
    divide(mid + 1, r);
}
```

1-14 扩展gcd

```
int exgcd(int a, int b, int &x, int &y) {
    if (!b) return (x = 1, y = 0, a);
    int g = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return g;
}
```

对于方程 $ax+by=c$, 调用 **exgcd **, 求出 x_0 和 y_0 使得 $ax_0+by_0=\gcd(a, b)$

则在 $\gcd(a,b)\mid c$ 的情况下有通解 $$ x = x_0 \times \frac{c}{\gcd(a, b)}+k\times \frac{b}{\gcd(a, b)} \ y = y_0 \times \frac{c}{\gcd(a, b)}-k\times \frac{a}{\gcd(a, b)} $$

1-15 扩展crt(中国剩余定理)

```
//n个方程 x ≡ a (mod m)  解出最小的x，无解输出-1
//需要配合exgcd()
int excrt(const VI &a, const VI &m) {
    int x = 0, M = 1, n = a.size();
    bool ok = true;
    for (int i = 0; i < n; i++) {
        int r = a[i] - x, j, y;
        r %= m[i];
        if (r < 0) r += m[i];
        int g = exgcd(M, m[i], j, y);
        if (r % g) ok = false;
        j = ((__int128)j * r / g % (m[i] / g));
        if (j < 0) j += m[i] / g;
        int lM = M;
        M = M / g * m[i]; // M=lcm(M,m[i])
        x = ((__int128)j * lM % M + x) % M;
    }
    if (ok) return x;
    else return -1;
}
```

1-16 高斯消元

```
const ld eps = 1e-9;
const int N = 128;
int gauss(ld a[][N], int n) {
    int c, r;
    for (c = 0, r = 0; c < n; ++c) { // c列r行，遍历列
        int t = r;
        for (int i = r; i < n; ++i) // 寻找列主元，拿t记录
            if (std::fabs(a[i][c]) > std::fabs(a[t][c]))
                t = i;
        if (std::fabs(a[t][c]) < eps)
            continue; // 如果列主元为0，不必考虑，当前列全为0
        // 交换列主元t行与当前r行的数
        for (int i = c; i < n + 1; ++i)
            std::swap(a[t][i], a[r][i]);
        // 当前列主元已经被交换到了r行，需要从后向前进行处理，避免主对角线元素变成1
        for (int i = n; i >= c; --i)
            a[r][i] /= a[r][c];
        // 列主消元
        for (int i = r + 1; i < n; ++i)
            if (std::fabs(a[i][c]) > eps)
                for (int j = n; j >= c; --j)
                    a[i][j] -= a[r][j] * a[i][c];
        ++r;
    }
    if (r < n) {
        for (int i = r; i < n; ++i)
            if (std::fabs(a[i][n]) > eps)
                return 2; // 2 则无解
        return 1;          // 1 无穷多解
    }
    // 上三角阶梯型矩阵
    // 直接求解即可，最后一列放置结果
    for (int i = n - 1; i >= 0; --i)
        for (int j = i + 1; j < n; ++j)
            a[i][n] -= a[j][n] * a[i][j];
    return 0;
}
```

1-17 线性基

```
i64 b[64], flag;
// 线性基数组 , flag 表示线性基能不能异或出 0
bool insert(i64 x) { // 插入一个数到线性基里
    for (int i = 62; i >= 0; i--)
        if (x >> i & 1) {
            if (!b[i]) // 线性基里没有第i位的项
                return b[i] = x, true;
            x ^= b[i]; // 线性基里有,则异或掉第i位
        }
    flag = true;
    return false;
}
bool check(i64 x) { // 询问线性基能不能异或出 x
    for (int i = 62; i >= 0; i--)
        if (x >> i & 1) {
            if (!b[i]) // 线性基里没有第i位的项
                return false;
            x ^= b[i]; // 线性基里有,则异或掉第i位
        }
    return true;
}
```

1-18 前缀线性基

```
// 题目大概率要开long long
int b[N][20], pos[N][20]; // 20 是 std::__lg(max_value)
void insert(int i, int x) {
    // 插入第 i 项数为 x
    memcpy(b[i], b[i - 1], sizeof b[i]); // 一定要从左到右顺着插入
    memcpy(pos[i], pos[i - 1], sizeof pos[i]);
    for (int j = 19, w = i; j >= 0; j--)
```

```
        if (x >> j & 1) {
            if (!b[i][j]) {
                b[i][j] = x, pos[i][j] = w;
                return;
            }
            if (pos[i][j] < w)
                std::swap(pos[i][j], w), std::swap(b[i][j], x);
            x ^= b[i][j];
        }
}
int ask(int l, int r) { // 询问区间[l, r]的最大异或值
    // 如果问第 k 小异或值记得要考虑 0 的结果
    int ans = 0;
    for (int j = 19; j >= 0; j--)
        if (l <= pos[r][j])
            ans = std::max(ans, ans ^ b[r][j]);
    return ans;
}
```

1-19 组合数公式

$$ C_{n}^{m-1} +C_{n}^{m} = C_{n+1}^m \ \ m\times C_n^m = n\times C_{n-1}^{m-1} \ \ 1\times C_n^1 + 2\times C_n^2 + 3\times C_n^3 +...+n\times C_n^n = n\times 2^{n-1} \ (由上一条定理易得) \ \ 1^2\times C_n^1 + 2^2\times C_n^2 + 3^2\times C_n^3 +...+n^2\times C_n^n = n(n+1)\times 2^{n-2} \(仿照上一条定理证明过程, 记得把i拆成i-1+1再分开两项可证) \ \ C_k^k+C_{k+1}^k+C_{k+2}^k+...+C_n^k=C_{n+1}^{k+1} \(画出杨辉三角易得) \ \ \sum_{i=0}^n (C_n^i)^2 = C_{2n}^n \(左边n个红球,右边n个蓝球,一共取n个球, 有几种取法) $$

1-20 二项式反演

$$ f(n) = \sum_{i=0}^{n}g(i) \ \ \ \Leftrightarrow \ \ \ g(n)=\sum_{i=0}^{n}(-1)^{n-i}×C_n^i×f(i) \ f(n) = \sum_{i=n}^{m}g(i) \ \ \ \Leftrightarrow \ \ \ g(n)=\sum_{i=n}^{m}(-1)^{n-i}×C_i^n×f(i) $$

完全积性函数反演

image-20231009170044242

1-21 容斥原理

image-20230404003115683

1-22 FWT

```
void add(int &a, const int &b) {
    a += b;
    a += (a < 0 ? p : (a >= p ? -p : 0));
}
void fwt(int *f, int n, int op, void (*conv)(int &, int &, int)) {
    for (int a = 1; a < n; a <<= 1)
        for (int b = a << 1, i = 0; i < n; i += b)
```

```
            for (int j = i; j < i + a; j++)
                conv(f[j], f[j + a], op);
}
void opor(int &a, int &b, int op) {
    add(b, op * a);
}
void opand(int &a, int &b, int op) {
    add(a, op * b);
}
const int inv2 = p - p / 2;
void opxor(int &a, int &b, int op) {
    auto x = a, y = b;
    a = 1ll * (x + y) % p * (op == 1 ? 1 : inv2) % p;
    b = 1ll * (x - y + p) % p * (op == 1 ? 1 : inv2) % p;
}
```

1-23 子集卷积

image-20230404091131341

```
void add(int &a, const int &b) {
    a += b;
    a += (a < 0 ? p : (a >= p ? -p : 0));
}
void fwt(int *f, int n, int op) {
    for (int j = 0; j < n; j++)
        for (int i = 0; i < 1 << n; i++)
            if (i >> j & 1)
                add(f[i], op * f[i ^ (1 << j)]);
}
int count(int x) {
    int res = 0;
    while (x) x -= x & -x, res++;
    return res;
}
int f[21][N], g[21][N], h[21][N];
// c[i] =  sum  a[j]*b[k]
//    (j|k=i j&k=0)
void subsetConv(int *a, int *b, int *c, int n) {
    // input a and b , ouput c, len = 1 << n
    for (int i = 0; i < 1 << n; i++)
        f[count(i)][i] = a[i], g[count(i)][i] = b[i];
    for (int i = 0; i <= n; i++)
        fwt(f[i], n, 1), fwt(g[i], n, 1);
    for (int i = 0; i <= n; i++)
        for (int j = 0; j <= n; j++)
            if (i + j <= n)
                for (int k = 0; k < 1 << n; k++)
                    add(h[i + j][k], 1ll * f[i][k] * g[j][k] % p);
    for (int i = 0; i <= n; i++)
        fwt(h[i], n, -1);
    for (int i = 0; i < 1 << n; i++)
        c[i] = h[count(i)][i];
}
```

复杂度：$O(n^2×2^n)$ n是bit数

有趣的知识

python 的自我打印程序

```
_='_=%r;print(_%%_)';print(_%_)
```

2, 动态规划

2-1 多重背包

```
std::vector<i64> multiBag(std::vector<std::array<int, 3>>& goods, int S) {
    // S 总背包大小
    std::vector<i64> f(S + 1, 0);
    std::vector<int> q(S + 2, 0);
    for (auto [v, w, m] : goods) {
        // v价值, w体积, m数量
        auto calc = [&](int i, int j) {
            return f[j] + 1ll * (i - j) / w * v;
        };
        for (int up = S; up + w > S; up--) {
            int l = 1, r = 0, k = up;
            for (int x = up; x > 0; x -= w) {
                for (; k >= std::max(0ll, x - 1ll * m * w); k -= w) {
                    while (l <= r and calc(x, k) > calc(x, q[r]))
                        r--;
                    q[++r] = k;
                }
                f[x] = calc(x, q[l]);
                if (q[l] == x)
                    l++;
            }
        }
    }
    return f;
}
```

2-2 基数排序

```
const int B = 10;
void radix_sort(int a[], int n) { // 下标从0开始
    int *b = new int[n];          // 临时空间
    int *cnt = new int[1 << B];
    int mask = (1 << B) - 1;
    int *x = a, *y = b;
    for (int i = 0; i < 32; i += B) { // 对32位整数排序
        memset(cnt, 0, sizeof cnt);
        for (int j = 0; j < n; j++)
            ++cnt[x[j] >> i & mask];
        for (int sum = 0, j = 0; j < (1 << B); j++)
            sum += cnt[j], cnt[j] = sum - cnt[j];
        for (int j = 0; j < n; j++)
            y[cnt[x[j] >> i & mask]++] = x[j];
        std::swap(x, y);
    }
    delete[] cnt;
    delete[] b;
}
```

2-3 四边形不等式

对于 $dp[i][j] = \min_{i\le k\le j}(dp[i][k], dp[k+1][j]) + w[i][j]$ 的区间dp 。

不妨设在 $dp[i][j]$ 里取最优的 $k$ 为 $pos[i][j]$ 。

要求两个条件 ：

- 对于 $L\le l \le r\le R$ 有 $w[l][r] \le w[L][R]$
- 对于 $L\le l \le r\le R$ 有 $w[L][r]+w[l][R] \le w[l][r]+w[L][R]$

于是有这样的性质：

- $pos[i][j-1] \le pos[i][j] \le pos[i+1][j]$

于是得到了如下 $O(n^2)$ 的代码

```
for (int i = n - 1; i; i--)
    for (int j = i + 1; j <= n; j++)
        for (int k = pos[i][j - 1]; k <= pos[i + 1][j]; k++) {
            int update = f[i][k] + f[k + 1][j] + w(i, j);
            if (f[i][j] > update) {
                f[i][j] = update;
                pos[i][j] = k;
            }
        }
```

- 注意 $\max$ 函数慎用这个

3, 字符串

3-1 快读

```
//超快的快读
#define getchar() (tt == ss && (tt = (ss = In) + fread(In, 1, 1 << 20, stdin), ss
== tt) ? EOF : *ss++)
char In[1 << 20], *ss = In, *tt = In;
inline void read(int &res) {
    bool fu = 0;
    res = 0;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-')
            fu = 1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9')
        res = (res << 3) + (res << 1) + ch - '0', ch = getchar();
    if (fu)
        res = -res;
}
inline std::istream &operator>(std::istream &in, int &a) {
    read(a);
    return in;
}
//用法 cin>n;
```

3-2 快写

```
void write(__int128 x) {
    if (x < 0)
        putchar('-'), x = -x;
    if (x > 9)
        write(x / 10); // 递归输出
    putchar(x % 10 + '0');
}
```

3-3 Kmp

```
void kmp_0(char *s, int *link, int n) {
    // index from 0
    std::fill(link, link + n + 1, 0);
    for (int i = 1, j = 0; i < n; i++) {
        while (j and s[i] != s[j])
            j = link[j - 1];
        j += (s[i] == s[j]);
        link[i] = j;
    }
}
void match_0(char *t, int n, char *S, int m, int *link) {
    // index from 0
    for (int i = 0, j = 0; i < m; i++) {
        while (j and S[i] != t[j])
            j = link[j - 1];
        j += (S[i] == t[j]);
        if (j == n) {
            // is matched
            j = link[j - 1];
        }
    }
}
void kmp_1(char *s, int *link, int n) {
    // index from 1
    std::fill(link, link + n + 1, 0);
    for (int i = 2, j = 0; i <= n; i++) {
        while (j and s[i] != s[j + 1])
            j = link[j];
        j += (s[i] == s[j + 1]);
        link[i] = j;
    }
}
void match_1(char *t, int n, char *S, int m, int *link) {
    // index from 1
    for (int i = 1, j = 0; i <= m; i++) {
        while (j and S[i] != t[j + 1])
            j = link[j];
        j += (S[i] == t[j + 1]);
        if (j == n) {
            // is matched
            j = link[j];
        }
    }
}
```

3-4 Z函数

```
int z[N];
void Zalgo(char *t, int tlen, char *s, int slen, int *p) {
    // index from 0
    std::fill(z, z + tlen + 1, 0); // clear
    for (int i = 1, l = 0, r = 0; i < tlen; i++) {
        if (i < r)
            z[i] = std::min(z[i - l], r - i);
        while (i + z[i] < tlen and t[i + z[i]] == t[z[i]])
            z[i]++;
        if (i + z[i] > r)
            l = i, r = i + z[i];
    }
    // z[i] 表示t[i,tlen-1]和t的最长公共前缀lcp的长度
    for (int i = 0, l = 0, r = 0; i < slen; i++) {
        if (i < r)
            p[i] = std::min(z[i - l], r - i);
        while (i + p[i] < slen and p[i] < tlen and s[i + p[i]] == t[p[i]])
            p[i]++;
        if (i + p[i] > r)
            l = i, r = i + p[i];
    }
    // p[i] 表示s[i,slen-1]和t的最长公共前缀lcp的长度
}
```

$O(n)$

3-5 马拉车

```
void manacher(char *s, int n, int *r, int *f) {
    // input with index-1
    s[0] = '[', s[n + 1 << 1] = ']';
    for (int i = n << 1 | 1, j = n; i; i--)
        s[i] = (i % 2 ? '-' : s[j--]);
    int mid = 1, far = 1;
    std::fill(f, f + n * 2 + 3, 0); // initial f
    for (int i = 1; i <= (n << 1 | 1); i++) {
        r[i] = std::min(r[2 * mid - i], far - i);
        // 要是题目说s[i]和s[i]不匹配了这里设 r[i] = 0
        while (s[i + r[i]] == s[i - r[i]])
            r[i]++;
        if (far < i + r[i])
            mid = i, far = i + r[i];
        f[i + r[i] - 1] = std::max(f[i + r[i] - 1], r[i]);
    }
    for (int i = n << 1; i; i--)
        f[i] = std::max(f[i], f[i + 1] - 1);
    // f[i << 1] 就是以第 i 个字符结尾的最长回文子串
}
// 变换使得 s[i << 1] 就是原本的 s[i]
// r[i] - 1 是以 i 为中心的回文子串的真实长度
```

3-6 AC自动机

```
char s[N], t[128];
int n, m, tot, slen, tlen;
int ed[N >> 2], ht[N >> 2], link[N >> 2], son[N >> 2][26];
bool f[N];
unsigned int mk[N >> 2];

void insert() {
    int now = 0;
    for (int i = 1; i <= tlen; i++) {
        if (!son[now][t[i] - 'a']) {
            son[now][t[i] - 'a'] = ++tot;
        }
        now = son[now][t[i] - 'a'];
    }
    ed[now] = tlen;
    mk[now] = 1 << tlen;
}
void build() {
    queue<int> q;
    for (int i = 0; i < 26; i++) {
        if (son[0][i]) {
            q.push(son[0][i]);
        }
    }
    while (!q.empty()) {
        int i = q.front();
        q.pop();
        mk[i] = mk[i] | mk[link[i]];
        int k = 0;
        for (int &j : son[i]) {
            if (j) {
                link[j] = son[link[i]][k];
                q.push(j);
            } else
                j = son[link[i]][k];
            k++;
        }
    }
}

int signed main() {
    IOS;
    cin >> n >> m;
    while (n--) {
        cin >> (t + 1);
        tlen = strlen(t + 1);
        insert();
    }
    build();
    while (m--) {
        cin >> (s + 1);
        slen = strlen(s + 1);
        f[0] = true;
        for (int i = 1; i <= slen; i++)
            f[i] = false;
        int now = 0;
        unsigned int tmp = 1;
        for (int i = 1; i <= slen; i++) {
            now = son[now][s[i] - 'a'];
            //正常查询应该是 for(int j=now;j;j=link[j])
            if (mk[now] & (tmp <<= 1))
                f[i] = true, tmp |= 1;
        }
        while (!f[slen])
            slen--;
        cout << slen << '\n';
    }
}
```

3-7 后缀数组

```
const int N = 2e5+20;
// N是字符串的长度
struct SA {
    std::string s;
    // key1[i] = rk[id[i]]（作为基数排序的第一关键字数组）
```

```
int n, sa[N], rk[N], ork[N << 1], id[N], key1[N], cnt[N], ht[N];
int st[N][20];
// sa[i]是排名为i的后缀的下标
// rk[i]是下标为i的后缀的排名
bool cmp(int x, int y, int w) {
    return ork[x] == ork[y] && ork[x + w] == ork[y + w];
}
void build() {
    n = s.size();
    s = "0" + s;
    // s串下标从1开始
    int m = 127;
    for (int i = 1; i <= n; i++)
        cnt[rk[i] = s[i]]++;
    for (int i = 1; i <= m; i++)
        cnt[i] += cnt[i - 1];
    for (int i = n; i >= 1; i--)
        sa[cnt[rk[i]]--] = i;

    for (int w = 1, u;; w <<= 1, m = u) { // m=u 就是优化计数排序值域
        u = 0;
        for (int i = n; i > n - w; i--)
            id[++u] = i;
        for (int i = 1; i <= n; i++)
            if (sa[i] > w)
                id[++u] = sa[i] - w;
        memset(cnt, 0, sizeof(cnt));
        //或者也可以 for(int i=0;i<=n+m;i++)cnt[i]=0;
        for (int i = 1; i <= n; ++i)
            cnt[key1[i] = rk[id[i]]]++;
        // 注意这里px[i] != i，因为rk没有更新，是上一轮的排名数组

        for (int i = 1; i <= m; i++)
            cnt[i] += cnt[i - 1];
        for (int i = n; i >= 1; i--)
            sa[cnt[key1[i]]--] = id[i];
        for (int i = 1; i <= n; i++)
            ork[i] = rk[i];
        u = 0;
        for (int i = 1; i <= n; i++)
            rk[sa[i]] = cmp(sa[i], sa[i - 1], w) ? u : ++u;
        if (u == n)
            break;
    }
    //这里往上就是求sa数组
    for (int i = 1, k = 0; i <= n; ++i) {
        if (rk[i] == 0)
            continue;
        if (k)
            k--;
        while (s[i + k] == s[sa[rk[i] - 1] + k])
            k++;
        ht[rk[i]] = k;
    }
```

```
        //这里是求ht数组
        // ht[i]=lcp(sa[i],sa[i-1])
        // lcp(i,j)是后缀i和后缀j的最长公共前缀
        for (int i = 1; i <= n; i++)
            st[i][0] = ht[i];
        for (int j = 1; j < 20; j++)
            for (int i = 1; i <= n; i++)
                if (i + (1 << (j - 1)) <= n)
                    st[i][j] = std::min(st[i][j - 1], st[i + (1 << (j - 1))][j -
1]);
    }
    int getmin(int i, int j) {
        int k = std::__lg(j - i + 1);
        return std::min(st[i][k], st[j - (1 << k) + 1][k]);
    }
} pre, pot;
```

复杂度: $O(nlogn)$

3-8 后缀自动机

```
char s[N]; //cin>>(s+1)
const int ZF=26;//字符集大小，太大就开map，或考虑后缀数组
int son[N<<1][ZF],len[N<<1],link[N<<1]={-1};
int tot, last; //tot是最大节点(inclusive)
//int cnt[N<<1]; 计算i节点子串出现次数
void add(char c) {
    c-='a';
    int cur = ++tot;
    // 如果要统计子串出现次数 cnt[cur]++;
    len[cur] = len[last] + 1;
    int v = last;
    while (v != -1  and !son[v][c]) {
        son[v][c] = cur;
        v = link[v];
    }
    if (v == -1) link[cur] = 0;
    else {
        int q = son[v][c];
        if (len[v] + 1 == len[q]) link[cur] = q;
        else {
            int clone = ++tot;
            len[clone] = len[v] + 1;
            for(int i = 0; i < ZF; i++)
                son[clone][i]=son[q][i];
            link[clone] = link[q];
            while (v != -1 and son[v][c] == q) {
                son[v][c] = clone;
                v = link[v];
            }
            link[q] = link[cur] = clone;
        }
    }
    last = cur;
}
```

SAM 压内存 $O(n\log n)$ 写法

```
char s[N];
int link[N << 1] = {-1}, len[N << 1];
std::map<std::pair<int, int>, int> son;
int cnt[N << 1], pos[N];
int last, tot;
void add(int id) {
    int c = s[id] - 'a';
    int cur = ++tot;
    // cnt[cur] = 1,  pos[id] = cur;
    len[cur] = len[last] + 1;
    int v = last;
    while (v != -1 and !son[{v, c}])
        son[{v, c}] = cur, v = link[v];
    if (v == -1)
        link[cur] = 0;
    else {
        int q = son[{v, c}];
        if (len[q] == len[v] + 1)
            link[cur] = q;
        else {
            int cl = ++tot;
            len[cl] = len[v] + 1;
            link[cl] = link[q];
            std::pair<int, int> p = {q, -1};
            auto it = son.lower_bound(p);
            while (it != son.end() and it->first.first == q) {
                son[{cl, it->first.second}] = it->second;
                it = son.upper_bound(it->first);
            }
            while (v != -1 and son.count({v, c}) and son[{v, c}] == q)
                son[{v, c}] = cl, v = link[v];
            link[cur] = link[q] = cl;
        }
    }
    last = cur;
}
```

3-9 广义后缀自动机（在线）

```
char s[N];
int len[N << 1], link[N << 1] = {-1}, son[N << 1][26];
int tot, last;
void exadd(char c) {
    c -= 'a';
    if (son[last][c]) {
        int v = last, q = son[v][c];
        if (len[q] != len[v] + 1) {
            int cl = ++tot;
            len[cl] = len[v] + 1;
            link[cl] = link[q];
            for (int i = 0; i < 26; i++)
                son[cl][i] = son[q][i];
            while (v != -1 and son[v][c] == q)
                son[v][c] = cl, v = link[v];
            link[q] = cl;
            q = cl;
        }
        int cur = last = q;
        // 可以在这里 对cur点加信息
        return;
    }

    int cur = ++tot;
    len[cur] = len[last] + 1;
    //在这里对 cur点加信息
    int v = last;
    while (v != -1 and son[v][c] == 0)
        son[v][c] = cur, v = link[v];
    if (v == -1)
        link[cur] = 0;
    else {
        int q = son[v][c];
        if (len[q] == len[v] + 1)
            link[cur] = q;
        else {
            int cl = ++tot;
            len[cl] = len[v] + 1;
            link[cl] = link[q];
            for (int i = 0; i < 26; i++)
                son[cl][i] = son[q][i];
            while (v != -1 and son[v][c] == q)
                son[v][c] = cl, v = link[v];
            link[q] = link[cur] = cl;
        }
    }
    last = cur;
}
```

3-10 广义后缀自动机(离线)

```
char s[N];
int len[N << 1], link[N << 1] = {-1}, son[N << 1][26];
int tot;
int exadd(int last, int c) { // last and nowchar
    int cur = son[last][c];
    len[cur] = len[last] + 1;
    int v = link[last]; //如果没有这一步 下一个 while 会直接break
    while (v != -1 and son[v][c] == 0)
        son[v][c] = cur, v = link[v];
    if (v == -1)
        link[cur] = 0;
    else {
        int q = son[v][c];
        if (len[q] == len[v] + 1)
            link[cur] = q;
        else {
            int cl = ++tot; // clone to q
            len[cl] = len[v] + 1;
            link[cl] = link[q];
            for (int i = 0; i < 26; i++)
                son[cl][i] = (len[son[q][i]] ? son[q][i] : 0); //这里和sam不一样
            while (v != -1 and son[v][c] == q)
                son[v][c] = cl, v = link[v];
            link[cur] = link[q] = cl;
        }
    }
    return cur;
}
void trie() {
    int slen = strlen(s + 1);
    int p = 0;
    for (int j = 1; j <= slen; j++) {
        if (son[p][s[j] - 'a'] == 0)
            son[p][s[j] - 'a'] = ++tot;
        p = son[p][s[j] - 'a'];
        // 如果有需要可以在这个位置进行线段树动态开点
        // root[p] = tradd(root[p], 1, n, id);
    }
}
void buildExSam() {
    std::queue<std::pair<int, int>> q;
    for (int i = 0; i < 26; i++)
        if (son[0][i])
            q.push({0, i}); // last and nowchar
    while (!q.empty()) {
        auto p = q.front();
        q.pop();
        int last = exadd(p.first, p.second);
        for (int i = 0; i < 26; i++)
```

```
            if (son[last][i])
                q.push({last, i});
    }
}
// 一个性质好像是trie()之后build()之前，1-tot都是endpos
// std::cin >> n; // 输入字符串个数
// for (int i = 1; i <= n; i++)
//     std::cin >> (s + 1), trie();
// buildExSam();
```

3-11 最小表示法

```
std::vector<int> minShift(std::vector<int> &a) {
    int n = a.size();
    int i = 0, j = 1, k = 0;
    while (k < n and i < n and j < n) {
        if (a[(i + k) % n] == a[(j + k) % n])
            k++;
        else {
            (a[(i + k) % n] > a[(j + k) % n] ? i : j) += k + 1;
            i += (i == j);
            k = 0;
        }
    }
    k = std::min(i, j);
    std::vector<int> ans(n);
    for (int i = 0; i < n; i++)
        ans[i] = a[(i + k) % n];
    return ans;
}
// 直接返回字典序最小循环同构串
```

4, 数据结构

4-1 动态bitset

```
template <int LEN>
void solve(int m) { // 要开 m 大小的bitset
    if (LEN < m)
        return solve<std::min(N, (LEN << 1))>(m);
    std::bitset<LEN> f;
    //
}
// 用法  solve<1>(m)
```

4-2 带权并查集

```cpp
int find(int x) {
    if (f[x] == x)
        return x;
    int F = f[x];
    int res = find(f[x]);
    // g[x] <- g[F] 看情况把原父亲的信息压缩到x上
    // 比如 g[x] ^= g[F];
    return f[x] = res;
}

void Union(int u, int v, int w) {
    int fu = find(u), fv = find(v);
    if (fu == fv)
        return;
    // g[fv] <- (g[u], g[v], w)
    // 四边形将 v->u 转移到 fv->fu
    // 比如 g[fv] = w ^ g[u] ^ g[v];
    f[fv] = fu;
}
```

4-3 可删堆

```cpp
template <class T, class cmp = std::less<T>>
struct heap {
    std::priority_queue<T, std::vector<T>, cmp> A, B; // heap=A-B

    void push(T x) { A.push(x); }

    void erase(T x) { B.push(x); }

    T top() {
        while (!B.empty() && A.top() == B.top())
            A.pop(), B.pop();
        return A.top();
    }

    void pop() {
        while (!B.empty() && A.top() == B.top())
            A.pop(), B.pop();
        A.pop();
    }

    int size() { return A.size() - B.size(); }
};
```

4-4 树哈希

有根树哈希 : $hash(x)$ 表示以 x 为根的子树的哈希值, $f( )$ 是随机函数 $$ hash(x) = 1+\sum_{y\ is \ son \ of \ x} f(hash(y)) $$ 无根树哈希则取树重心即可, 或者换根dp得到所有点为根的$hash$值, f 函数如下

```
using u64 = unsigned long long;
u64 f(u64 x) {
    auto h = [&](u64 y) {
        return (u64)y * y * y * 1237123 + 19260817;
    };
    return h(x & ((1ll << 31) - 1)) + h(x >> 31);
}
```

4-5 区间嵌套树

```
#define pii std::pair<int, int>
#define l first
#define r second
int n, m;
pii seg[N]; // 区间数组,建树以下标索引区间
std::vector<int> e[N]; // 树
signed main() {
    std::cin >> n >> m;
    for (int l, r, i = 1; i <= m; i++) {
        std::cin >> l >> r; // 输入
        seg[i] = (pii){l, r};
    }
    seg[0] = {1, n}; // 全局区间
    std::sort(seg, seg + m + 1); // 大概率题目是需要去重的
    m = std::unique(seg, seg + m + 1) - (seg + 1);
    std::sort(seg, seg + m + 1, [&](const pii &a, const pii &b) {
        return a.l == b.l ? a.r > b.r : a.l < b.l;
    }); // 排序
    std::vector<int> q(1, 0); // 栈, 里面初始元素是全局区间
    for (int i = 1; i <= m; i++) {
        while (seg[q.back()].r < seg[i].l) // 找到第一个包含seg[i]的区间
            q.pop_back();
        e[q.back()].push_back(i);
        q.push_back(i);
    }
    // 这样一个区间嵌套树就建出来了
}
```

4-6 圆嵌套树

```
i64 pf(int a) { return 1ll * a * a; }
const int N = 2e5 + 20;
const ld eps = 1e-8;
```

```cpp
struct Circle {
    int x, y, r;
} c[N];
struct event {
    int id;
    char op;
    bool operator<(const event &a) const {
        int u = c[id].x + (op == 'L' ? -c[id].r : c[id].r);
        int v = c[a.id].x + (a.op == 'L' ? -c[a.id].r : c[a.id].r);
        return u < v;
    }
} e[N << 1];
struct Semi {
    static int X;
    int id;
    char half;
    Semi(int jd, char h) : id(jd), half(h) {}
    bool operator<(const Semi &a) const { // 一定要const
        ld u = c[id].y + (half == 'u' ? 1 : -1) * sqrtl(pf(c[id].r) - pf(X -
c[id].x) + eps); // + eps 很重要
        ld v = c[a.id].y + (a.half == 'u' ? 1 : -1) * sqrtl(pf(c[a.id].r) - pf(X -
c[a.id].x) + eps); // + eps 很重要
        return u < v;
    }
};
int Semi::X = 0;
std::vector<int> g[N];
int link[N];

signed main() {
    int n;
    std::cin >> n;
    for (int i = 1; i <= n; i++) {
        std::cin >> c[i].x >> c[i].y >> c[i].r;
        e[i * 2 - 1].id = e[i * 2].id = i;
        e[i * 2 - 1].op = 'L', e[i * 2].op = 'R';
        // 拆成上半圆和下半圆
    }
    std::sort(e + 1, e + 2 * n + 1);
    std::set<Semi> set;
    for (int i = 1; i <= 2 * n; i++) {
        Semi::X = c[e[i].id].x + (e[i].op == 'L' ? -c[e[i].id].r : c[e[i].id].r);
        // 扫描线
        if (e[i].op == 'R') {
            set.erase((Semi){e[i].id, 'u'});
            set.erase((Semi){e[i].id, 'd'});
        } else {
            set.insert((Semi){e[i].id, 'u'});
            set.insert((Semi){e[i].id, 'd'});
            auto it = set.upper_bound((Semi){e[i].id, 'u'});
            if (it == set.end())
                continue;
            link[e[i].id] = (it->half == 'u' ? it->id : link[it->id]);
        }
    }
    for (int i = 1; i <= n; i++)
        g[link[i]].push_back(i);
    // 这样 g 图就是一个圆的包含树了, 0 是 虚空大圆
}
```

4-7 点分治

树重心

```
//------------------------get root--------------------------
int mk[N], size[N], maxs[N];
int root;
void getRoot(int x, int fa, int tot) {
    size[x] = 1;
    maxs[x] = 0; // 表示 x 为根的子树里的最大子树的size
    for (auto [y, w] : e[x]) {
        if (y == fa or mk[y])
            continue;
        getRoot(y, x, tot);
        size[x] += size[y];
        maxs[x] = std::max(maxs[x], size[y]);
    }
    maxs[x] = std::max(maxs[x], tot - size[x]);
    if (root == -1 or maxs[x] < maxs[root])
        root = x;
}

//-----------计算过x点的路径的情况-----------------------
void getDis(int x, int fa, int len) {
    // arr2 <- len
    for (auto [y, w] : e[x]) {
        if (y == fa or mk[y])
            continue;
        getDis(y, x, len + w);
    }
}
void calc(int x) {
    // clear arr1 + 加入根信息
    for (auto [y, w] : e[x]) {
        if (mk[y])
            continue;
        // clear arr2
        getDis(y, x, w); // update arr2
        // 计算答案 ans <- arr1 + arr2
        // 更新容器 arr1 <- arr2
    }
}
//------------x就是当前重心，然后去分治-----------------
void divide(int x) {
    mk[x] = 1;
    calc(x);
    for (auto [y, w] : e[x]) {
        if (mk[y])
            continue;
        root = -1;
        getRoot(y, x, size[y]);
        getRoot(root, -1, size[y]);
        divide(root);
    }
}
signed main() {
    std::ios_base::sync_with_stdio(0), std::cin.tie(0);

    std::cin >> n;
    for (int u, v, w, i = 1; i < n; i++) {
        std::cin >> u >> v >> w;
        e[u].push_back({v, w});
        e[v].push_back({u, w});
    }
    // input query
    root = -1;
    getRoot(1, -1, n);
    getRoot(root, -1, n);
    divide(root);
    // output answer
}
```

4-8 树状数组

4-8-1 区间修改+区间查询

```
template <class T>
struct BIT {
    std::vector<T> t[2];
    BIT(int n) {
        t[0].assign(n, 0);
        t[1].assign(n, 0);
    }

    void add(int l, int r, T x) {
        if (l <= 0)
            for (; r < t[0].size(); r += r & -r)
                t[-l][r] += x;
        else {
            add(0, l, x);
            add(0, r + 1, -x);
            add(-1, l, l * x);
            add(-1, r + 1, -(r + 1) * x);
        }
    }
    T get(int l, int r) {
        T res = 0;
        if (l <= 0)
            for (; r; r -= r & -r)
                res += t[-l][r];
        else {
            res = (r + 1) * get(0, r) - get(-1, r);
            res -= l * get(0, l - 1) - get(-1, l - 1);
        }
        return res;
    }
};
```

4-8-2 二维,子矩阵加&子矩阵求和

```
i64 t1[N][N], t2[N][N], t3[N][N], t4[N][N];

void update(int x, int y, int v) {
    for (int i = x; i <= n; i += i & -i)
        for (int j = y; j <= m; j += j & -j) {
            t1[i][j] += v;
            t2[i][j] += v * x;
            t3[i][j] += v * y;
            t4[i][j] += v * x * y;
        }
}

void add(int xa, int ya, int xb, int yb, int v) {
    //(xa, ya) 到 (xb, yb) 子矩阵
    update(xa, ya, v);
    update(xa, yb + 1, -v);
    update(xb + 1, ya, -v);
    update(xb + 1, yb + 1, v);
}

i64 sum(int x, int y) {
    i64 res = 0;
    for (int i = x; i; i -= i & -i)
        for (int j = y; j; j -= j & -j)
            res += (x + 1) * (y + 1) * t1[i][j] - (y + 1) * t2[i][j] -
                   (x + 1) * t3[i][j] + t4[i][j];
    return res;
}

i64 get(int xa, int ya, int xb, int yb) {
    //(xa, ya) 到 (xb, yb) 子矩阵
    return sum(xb, yb) - sum(xb, ya - 1) - sum(xa - 1, yb) + sum(xa - 1, ya - 1);
}
```

4-8-3 求区间最值

```cpp
// 单点修改，查最小值只能改小，查最大值只能改大
int b[N], t[N];
inline void modify(int i, int x) {
    b[i] = x;
    for (; i < N; i += i & -i)
        t[i] = std::min(t[i], x);
}
// 区间查询
inline int get(int l, int r) {
    int res = 1 << 30;
    while (l <= r) {
        res = std::min(res, b[r--]);
        for (; l <= r - (r & -r); r -= r & -r)
            res = std::min(res, t[r]);
    }
    return res;
}
```

4-9 树链剖分

```cpp
// modify 和 get 是线段树
std::vector<std::pair<int, int>> e[N];
int n, qn, tot;
int dfn[N], big[N], fa[N], size[N], dep[N], tp[N];
void dfs1(int i, int f) {
    fa[i] = f;
    dep[i] = dep[f] + 1;
    size[i] = 1;
    for (auto [j, w] : e[i])
        if (j != f) {
            dfs1(j, i);
            size[i] += size[j];
            if (size[j] > size[big[i]])
                big[i] = j;
        }
}
void dfs2(int i, int top) {
    tp[i] = top;
    dfn[i] = ++tot;
    if (big[i])
        dfs2(big[i], top);
    for (auto [j, w] : e[i]) {

        if (j != fa[i] and j != big[i])
            dfs2(j, j);
    }
}
```

```
int lca(int x, int y) {
    while (tp[x] != tp[y])
        (dep[tp[x]] > dep[tp[y]] ? x = fa[tp[x]] : y = fa[tp[y]]);
    return (dep[x] < dep[y] ? x : y);
}
int trget(int x, int y) {
    node ans[2];
    ans[0].s = ans[1].s = -1;
    int xy = lca(x, y);
    int j = 0;
    for (int i : {x, y}) {
        for (; tp[i] != tp[xy]; i = fa[tp[i]])
            ans[j] = get(1, 1, n, dfn[tp[i]], dfn[i]) + ans[j];
        ans[j] = get(1, 1, n, dfn[xy], dfn[i]) + ans[j];
        j++;
    }
    ans[0].reverse();
    auto res = ans[0] + ans[1];
    return res.s;
}
void trmodify(int x, int y, int c) {
    int xy = lca(x, y);
    for (int i : {x, y}) {
        for (; tp[i] != tp[xy]; i = fa[tp[i]])
            modify(1, 1, n, dfn[tp[i]], dfn[i], c);
        modify(1, 1, n, dfn[xy], dfn[i], c);
    }
}
```

4-10 主席树求区间第k小

```
int t[N * 80], ls[N * 80], rs[N * 80], root[N], tot;
// 动态开点 -> 左右儿子 + 空间开大 + tot
// t[i] 表示对应值域区间里有多少个数
int add(int j, int l, int r, int x) {
    // j 为第i-1棵值域线段树上的对应点
    // l, r 是值域区间
    int i = ++tot;
    if (l == r) {
        t[i] = t[j] + 1;
        return i;
    }
    ls[i] = ls[j], rs[i] = rs[j];
    int mid = l + r >> 1;
    if (x <= mid)
        ls[i] = add(ls[j], l, mid, x);
    else
        rs[i] = add(rs[j], mid + 1, r, x);
    t[i] = t[ls[i]] + t[rs[i]]; // up(i)
    return i;
}
```

```
int get(int j, int i, int l, int r, int k) {
    // l, r 是值域区间
    if (l == r)
        return l;
    int mid = l + r >> 1;
    int L = t[ls[i]] - t[ls[j]];
    if (L >= k) // 线段树上二分
        return get(ls[j], ls[i], l, mid, k);
    else
        return get(rs[j], rs[i], mid + 1, r, k - L);
}
```

4-11 树状数组套值域线段树

```
int root[N], t[N * 180], ls[N * 180], rs[N * 180], tot;

void modify(int &i, int l, int r, int x, int y) {
    // 第i个线段树里x的位置+y
    if (!i) i = ++tot;
    if (l == r) {
        t[i] += y;
        return;
    }
    int mid = l + r >> 1;
    if (x <= mid) modify(ls[i], l, mid, x, y);
    else modify(rs[i], mid + 1, r, x, y);
    t[i] = t[ls[i]] + t[rs[i]]; //up(i)
}
void add(int i, int x, int y) {
    for (; i < N; i += i & -i)
        modify(root[i], 1, m, x, y);
}
std::vector<int> po, ne; // 树状数组的两个前缀和相减 po - ne
// po 和 ne 里存的是线段树同一区间的节点
void get1(int l, int r) {
    po.clear(), ne.clear();
    for (l--; l; l -= l & -l)
        ne.push_back(root[l]);
    for (; r; r -= r & -r)
        po.push_back(root[r]);
}
int get2(int l, int r, int k) {
    if (l == r) return l;
    int mid = l + r >> 1;
    int sum = 0;
    for (int i : po) sum += t[ls[i]];
    for (int i : ne) sum -= t[ls[i]];
    if (k <= sum) {
        for (int &i : po) i = ls[i];
        for (int &i : ne) i = ls[i];
        return get2(l, mid, k);
    } else {
        for (int &i : po) i = rs[i];
        for (int &i : ne) i = rs[i];
        return get2(mid + 1, r, k - sum);
    }
}
```

4-12 线段树合并

```
int merge(int a, int b, int l, int r) {
  if (!a) return b;
  if (!b) return a;
  // 如果需要合并之后 还要区间查询或者单点查询
  // 就要np=++tot
  // return np
  // 而且空间要开很大

  if (l == r) {
    // do something of merge
    return a;
  }
  int mid = (l + r) >> 1;
  tr[a].l = merge(tr[a].l, tr[b].l, l, mid);
  tr[a].r = merge(tr[a].r, tr[b].r, mid + 1, r);
  pushup(a);
  return a;
}
```

4-13 李超线段树

```
const int LM = 0, RM = 1e9;
// 横坐标的左右边界, 注意视情况调节
struct line {
    ld k, b;             // 线段斜率, 截距
    int l = LM, r = RM; // 线段左右边界

    ld at(int x) { return k * x + b; }

    friend int cross(const line &a, const line &b) {
        return floor((a.b - b.b) / (b.k - a.k));
    }
};

struct node {
    line seg;
    node *ls, *rs;
};
```

```
struct LiChaoTree {
    LiChaoTree() { rt = new node(); }  // 一定要加括号初始化

    void insert(ld k, ld b, int lm = LM, int rm = RM) {
        line Z = (line){k, b, lm, rm};
        modify(rt, LM, RM, Z);
    }
    ld get(int x) {
        return query(rt, LM, RM, x);
    }
private:
    const ld eps = 1e-9;
    int sgn(const ld &a) { return (a < -eps ? -1 : a > eps); }
    node *rt;
    void modify(node *&p, int l, int r, line Z) {
        if (!p) // 一定要传node*&
            p = new node();  // 一定要加括号初始化!!

        int mid = l + r >> 1;
        if (Z.l <= l && r <= Z.r) {
            int L = sgn(Z.at(l) - p->seg.at(l));
            int R = sgn(Z.at(r) - p->seg.at(r));
            if (L + R == 2) { // 新线段完全在老线段之上
                p->seg = Z;
            } else if (L == 1 or R == 1) { // 新线段不完全在老线段之上
                if (sgn(Z.at(mid) - p->seg.at(mid)) > 0)
                    std::swap(p->seg, Z);

                if (sgn(cross(Z, p->seg) - mid) < 0)
                    modify(p->ls, l, mid, Z);
                else
                    modify(p->rs, mid + 1, r, Z);
            }
        } else {
            if (Z.l <= mid)
                modify(p->ls, l, mid, Z);
            if (mid < Z.r)
                modify(p->rs, mid + 1, r, Z);
        }
    }
    ld query(node *&p, int l, int r, int x) {
        if (!p)
            return 0;
        int mid = l + r >> 1;
        ld ans = p->seg.at(x);
        if (l < r)
            ans = std::max(ans, (x <= mid ? query(p->ls, l, mid, x) : query(p->rs,
mid + 1, r, x)));
        return ans;
    }
};
/*
用法 :
先调节 LM, RM
```

```
LiChaoTree t;
t.insert(斜率, 截距);  //插入直线
t.insert(斜率, 截距, 左x, 右x);  //插入线段
t.get(x);  //得到x处最大y
*/
```

4-14 pbds平衡树

```
#include <bits/extc++.h>
#define pb __gnu_pbds
// 这些都要求g++编译器, clang++不支持

pb::tree<T, pb::null_type, std::less<>, pb::rb_tree_tag,
pb::tree_order_statistics_node_update> t; // 红黑树
// 当set用,多下面两个函数, 注意其中会自动去重, 所以往往使用 T = std::pair<int, int>

t.order_of_key(x); // 查询有多少元素比x小
t.find_by_order(x); // 查询有x个元素比它小的元素的迭代器
```

4-15 平衡树Rope

头部

```
#include <bits/extc++.h>
using namespace __gnu_cxx;
```

创建

```
rope<int> rp;
// rope的下标是从 0 开始的
```

常用函数

```
rp.push_back(x); //尾部加上元素x
rp.replace(j, x);
// 把下标为 j 的元素换成 x ,如果 j ≥ rp.size() 会变成 push_back(x)

rp.insert(j, x);
// 在下标为 j 的元素左边, 下标为 j - 1 的元素的右边, 插入 x
// 如果 j ≥ rp.size() 会变成 push_back(x)

int a[N];
rp.insert(j, a, len);
```

```
// 在下标为 j 的元素左边, 下标为 j - 1 的元素的右边, 插入数组 a 的前 len 个数, len > N 会出问题
// 初始化的时候, 这样比每次 push_back 快

rp.erase(j, len); // 从下标 j 开始, 删除 len 个数, 没有 len 会直接删到结尾

rp.substr(j, len);
// 将下标为 j 开始的,长度为 len 的子段 取出来, 返回一个 rope<int>
// 如果 len 太大, 会出问题, 如果 j 太大, 返回空的 rope<int>

rp[j]; rp.at(j);
// 返回下标为 j 的元素, 但不是引用, 不能作为左值

rp = rp1 + rp2; // + 可以实现直接拼接

rope<int> *rp[N]; // 准备实现可持久化 -> 创建 N 个rope
rp[0] = new rope<int>(); // 初始版本
rp[i] = new rope<int>(*rp[v]); // 创建 v 版本的副本作为新版本 i 版本
rp[i] = new auto(*rp[v]); // 这样写也可以, 创建副本的操作是 O(1) 的
```

Rope 总体是 $O(N\sqrt N)$ 的

4-16 Rope实现可持久化并查集

```
const int N = 1e5 + 20;
rope<int> *rp[N];
int find(int i, int j) { //查询版本 i 的 find(j)
    int rpij = rp[i]->at(j);
    if (rpij == j)
        return j;
    int ans = find(i, rpij);
    if (ans != rpij) //这个 if 非常重要!
        rp[i]->replace(j, ans);
    return ans;
}
void Union(int i, int j, int k) { // 在版本 i 下合成set(j)和set(k)
    int fj = find(i, j), fk = find(i, k);
    if (fj == fk)
        return;
    rp[i]->replace(fk, fj);
}
```

4-17 ST表

```
template <class T, class CMP = std::less<T>>
struct RMQ {
    const CMP cmp = CMP();
    std::vector<std::vector<T>> ST;
    RMQ(const std::vector<T> &a) {
        int n = a.size(), logn = std::__lg(n);
        ST.assign(n, std::vector<T>(logn + 1));
        for (int i = 0; i < n; i++)
            ST[i][0] = a[i];
        for (int j = 0; j < logn; j++) {
            for (int i = 0; i + (1 << (j + 1)) - 1 < n; i++) {
                ST[i][j + 1] = std::min(ST[i][j], ST[i + (1 << j)][j], cmp);
            }
        }
    }

    T operator()(int l, int r) {
        int Log = std::__lg(r - l + 1);
        return std::min(ST[l][Log], ST[r - (1 << Log) + 1][Log], cmp);
    }
};
```

4-18 手写哈希表

```
using u64 = unsigned long long; // 有的时候用u32
const int mod = 1e6 + 3; // 自由调节
struct cpHash {
    int hd[mod], val[mod * 2], nt[mod * 2], tot;
    u64 to[mod * 2];
    void clear() {
        for (int i = 1; i <= tot; i++) {
            int u = to[i] % mod;
            hd[u] = 0;
        }
        tot = 0;
    }
    int &operator[](u64 x) {
        int u = x % mod;
        for (int i = hd[u]; i; i = nt[i])
            if (to[i] == x)
                return val[i];
        to[++tot] = x;
        nt[tot] = hd[u];
        hd[u] = tot;
        return val[tot] = 0;
    }
} mp;
```

5, 树论

5-1 树上启发式合并

```
//树上启发式合并
void dfs(int i,int fa){
//   先递归轻子树
//   if(当前节点有重子树){
//       递归重子树
//   }
//   加上所有轻子树贡献
//   加上当前节点贡献
//   统计当前节点答案
//   if(当前节点是自己父节点的重儿子)return
//   else 删除当前这棵树的贡献
}
```

5-2 树剖LCA

```
template <class T>
struct Lca {
    std::vector<std::vector<T>> &e;
    std::vector<int> size, big, dep, tp, fa;
    int n;
    Lca(std::vector<std::vector<T>> &g)
        : e(g), n(g.size() - 1), tp(g.size()), big(g.size()),
          size(g.size()), dep(g.size()), fa(g.size()) {
        dfs1(1, 0);
        dfs2(1, 1);
    }

    void dfs1(int x, int f) {
        dep[x] = dep[f] + 1;
        size[x] = 1;
        fa[x] = f;
        for (int y : e[x]) // 有权图 换成 auto [y, w]
            if (y != f) {
                dfs1(y, x);
                size[x] += size[y];
                if (size[y] > size[big[x]])
                    big[x] = y;
            }
    }
    void dfs2(int x, int top) {
        tp[x] = top;
        if (big[x])
            dfs2(big[x], top);
        for (int y : e[x]) // 有权图 换成 auto [y, w]
            if (y != big[x] and y != fa[x])
                dfs2(y, y);
    }

    int getLca(int x, int y) {
        while (tp[x] != tp[y])
```

```
            (dep[tp[x]] > dep[tp[y]] ? x = fa[tp[x]] : y = fa[tp[y]]);
        return (dep[x] < dep[y] ? x : y);
    }
    int dist(int x, int y) {
        int L = getLca(x, y);
        return dep[x] + dep[y] - 2 * dep[L];
    }
};
```

5-3 倍增LCA

```
int n, m, s, tot;
int lg; // lg=std::__lg(n)
std::vector<int> e[N];
int dep[N], f[N][19];
void dfs(const int& i,const int& fa) {
    for (int j : e[i]) {
        f[j][0] = i;
        dep[j] = dep[i] + 1;
        for (int k = 1; k <= lg; k++) {
            f[j][k] = f[f[j][k - 1]][k - 1];
            if (!f[j][k]) break;
        }
        dfs(to[j], i);
    }
}
int lca(int x, int y) {
    if (dep[x] > dep[y]) std::swap(x, y);
    for (int k = lg; k >= 0; k--)
        if (dep[f[y][k]] >= dep[x])
            y = f[y][k];
    if (x == y)
        return x;
    for (int k = lg; k >= 0; k--)
        if (f[x][k] != f[y][k])
            x = f[x][k], y = f[y][k];
    return f[x][0];
}
//注意这里lg=__lg(n)
```

复杂度： $O(nlogn)$预处理 $O(logn)$查询

6, 图论

6-1 单源最短路

6-1-1 spfa

复杂度 最坏 $O(nm)$

```
void spfa(int s) {
    memset(cnt, 0, sizeof cnt);
    dis[s] = 0;
    v[s] = 1;
    std::queue<int> q;
    q.push(s);
    while (!q.empty()) {
        int i = q.front();
        q.pop();
        v[i] = 0;
        for (auto [j, len] : e[i]) {
            if (dis[j] > dis[i] + len) {
                dis[j] = dis[i] + len;
                cnt[j] = cnt[i] + 1; //到to点的最短路边数
                if (cnt[j] >= n)
                    return; //边数大于n 说明有负环
                if (!v[j])
                    q.push(j), v[j] = 1;
            }
        }
    }
}
```

6-1-2 Dijkstra（非负权图）

复杂度 $O((n+m)\log n)$

```
auto dijkstra = [&](int s) {
    std::vector<i64> dis(n + 1, 1 << 30);

#define pii std::pair<i64, int>
    std::priority_queue<pii, std::vector<pii>, std::greater<>> q;

    dis[s] = 0;
    q.push({0, s});

    while (!q.empty()) {
        auto [D, x] = q.top();
        q.pop();
        if (D > dis[x])
            continue;

        for (auto [y, w] : e[x])
            if (dis[y] > dis[x] + w) {
                dis[y] == dis[x] + w;
                q.push({dis[y], y});
            }
    }

    return dis;
};
```

6-2 基环树预处理

```
std::vector<std::vector<int>> g;
std::vector<int> cycle;
void RingTree(std::vector<std::set<int>> &e) {
    int m = e.size(); // m = n + 1

    g.assign(m, {});
    cycle.clear();
    std::vector<int> vis(m, 0);

    std::function<void(int)> dfs1 = [&](int x) {
        vis[x] = 1;
        int y = *e[x].begin();
        e[x].erase(y);
        e[y].erase(x);
        g[y].push_back(x);
        if (e[y].size() == 1)
            dfs1(y);
    };
    for (int i = 1; i < m; i++)
        if (e[i].size() == 1)
            dfs1(i);

    std::function<void(int)> dfs2 = [&](int x) {
        cycle.push_back(x);
        vis[x] = 1;
        for (int y : e[x])
            if (!vis[y])
                dfs2(y);
    };

    for (int i = 1; i < m; i++)
        if (vis[i] == 0)
            dfs2(i);
}
```

6-3 最大流

```cpp
template <class T>
struct flow {
    struct E {
        int to;
        T cap;
        E(int to, T cap) : to(to), cap(cap) {}
    };

    const int n;
    std::vector<E> e;
    std::vector<std::vector<int>> g;
    std::vector<int> cur, h;

    flow(int n) : n(n), g(n) {}

    bool bfs(int s, int t) {
        h.assign(n, -1);
        std::queue<int> q;
        h[s] = 0;
        q.push(s);
        while (!q.empty()) {
            const int u = q.front();
            q.pop();
            for (int i : g[u]) {
                auto [v, c] = e[i];
                if (c > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    if (v == t)
                        return true;
                    q.push(v);
                }
            }
        }
        return false;
    }

    T dfs(int u, int t, T f) {
        if (u == t)
            return f;

        auto r = f;
        for (int &i = cur[u]; i < int(g[u].size()); ++i) {
            const int j = g[u][i];
            auto [v, c] = e[j];
            if (c > 0 && h[v] == h[u] + 1) {
                auto a = dfs(v, t, std::min(r, c));
                e[j].cap -= a;
                e[j ^ 1].cap += a;
                r -= a;
                if (r == 0)
```

```
                    return f;
                }
            }
            return f - r;
        }
        void add(int u, int v, T c) {
            g[u].push_back(e.size());
            e.emplace_back(v, c);
            g[v].push_back(e.size());
            e.emplace_back(u, 0);
        }
        T work(int s, int t) {
            T ans = 0;
            while (bfs(s, t)) {
                cur.assign(n, 0);
                ans += dfs(s, t, std::numeric_limits<T>::max());
            }
            return ans;
        }
    };

    // 用法
        // flow<int> f(n + 1);

        // for (int u, v, w, i = 0; i < m; i++) {
        //     std::cin >> u >> v >> w;
        //     f.add(u, v, w);
        // }

        // auto ans = f.work(s, t);
```

7, 计算几何

image-20230709160640534

7-1 基础部分

```
// 如果题目输出浮点数,还没有SPJ,务必检查会不会输出 -0.000的情况

const ld eps = 1e-9;
const ld pi = acosl(-1);
int sgn(const ld &a) {
    return (a < -eps ? -1 : a > eps);
}
#define pii std::pair<int,int>
#define x first
#define y second
pii operator+(const pii &a, const pii &b) {
    return {a.x + b.x, a.y + b.y};
}
pii operator-(const pii &a, const pii &b) {
    return {a.x - b.x, a.y - b.y};
}
i64 operator*(const pii &a, const pii &b) {
    return 1ll * a.x * b.x + 1ll * a.y * b.y;
}
pii operator*(const ld &a, const pii &b) {
    return {a * b.x, a * b.y};
}
i64 operator%(const pii &a, const pii &b) {
    return 1ll * a.x * b.y - 1ll * a.y * b.x;
}
pii rot(const pii &a, const ld &th) {
    // 逆时针旋转 th rad
    // 复数乘法 (a.x+a.yi)*(b.x+b.yi)=a.x*b.x-a.y*b.y+(a.x*b.y+a.y*b.x)i
    pii b = {cosl(th), sinl(th)};
    return {a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x};
}
```

7-2 点与点

```
i64 dis2(const pii &a, const pii &b = {0, 0}) {
    auto p = a - b;
    return p * p;
}
ld dis(const pii &a, const pii &b = {0, 0}) {
    auto p = a - b;
    return sqrtl(p * p);
}
```

7-3 点与线

```
bool onSeg(const pii &a, pii b, pii c) {
    // a on Seg b-c
    b = b - a, c = c - a;
    return (b % c == 0 and b * c <= 0);
}
ld disLine(const pii &a, const pii &b, const pii &c) {
    // a to line b-c
    auto v1 = b - c, v2 = a - c;
    ld ans = (ld)std::abs(v1 % v2) / sqrt(v1 * v1);
    return ans;
}
ld disSeg(const pii &a, const pii &b, const pii &c) {
    // a to seg b-c
    if ((a - b) * (c - b) <= 0 or (a - c) * (b - c) <= 0)
        return std::min(dis(a, b), dis(a, c));
    return disLine(a, b, c);
}
pii foot(const pii &a, const pii &b, const pii &c) {
    // 这里的 pii 是pair<ld, ld>
    // point a project to Line b-c
    auto u = a - b, v = c - b;
    ld rate = (u * v) / (v * v);
    return b + rate * v; // 有个数乘
}
pii symmetry(const pii &a, const pii &b, const pii &c) {
    // 这里的 pii 是pair<ld, ld>
    // point a symmetry to Line b-c
    auto ft = foot(a, b, c);
    return 2 * ft - a;
}
```

7-4 线与线

```
// 直线相交
pii cross(const pii &a, const pii &b, const pii &c, const pii &d) {
    // line a-b 交 line c-d
    // 这里的 pii 是 pair<ld, ld>
    auto v = c - d;
    ld sa = v % (a - d), sb = (b - d) % v;
    ld rate = sa / (sa + sb);
    return rate * (b - a) + a;
}
// 求向量夹角 tan
ld angle(const pii &a, const pii &b, const pii &c) {
    // a-b 和 a-c 的夹角tan 的绝对值
    auto v1 = b - a, v2 = c - a;
    if (v1 * v2 == 0)
        return 0; // 表示v1和v2垂直
    return (ld)(v1 % v2) / (v1 * v2);
}
```

7-5 求凸包

$O(n\log n)$

```
void convex(pii *p, int &n) { //大小为 n 的 pii 集合
    std::sort(p, p + n);      //为了取最左下角点 p[0]
    n = std::unique(p, p + n) - p;
    if (n <= 2) return; // 不写这个 会RE
    auto cmp = [&](const pii &a, const pii &b) {
        auto A = a - p[0], B = b - p[0];
        return (A % B == 0 ? dis2(A) < dis2(B) : A % B > 0);
    };
    std::sort(p + 1, p + n, cmp); //注意要 p+1
    std::vector<pii> qw(n + 1, p[0]);
    int r = 0;
    for (int i = 1; i < n; qw[++r] = p[i], i++)
        while (r and (qw[r] - qw[r - 1]) % (p[i] - qw[r]) <= 0)
            r--;
    for (int i = 0; i <= r; i++)
        p[i] = qw[i];
    n = r + 1; //一定要重设 n
    p[n] = p[0];
} // 把点集 p 改成凸包点集
```

7-6 点在不在多边形内

$O(n)$

```
bool onSeg(const pii &a, pii b, pii c) {
    // a on Seg b-c
    b = b - a, c = c - a;
    return (b % c == 0 and b * c <= 0);
}
int sgn(const i64 &a) {
    return (a < 0 ? -1 : a > 0);
}
int isCross(pii a, pii b, pii c, pii d) {
    // Seg c-d is Cross Seg a-b
    auto ab = b - a, cd = d - c;
    if (ab % cd == 0) return 0; // 共线判寄
    int r1 = sgn(ab % (c - a)), r2 = sgn(ab % (d - a));
    int t1 = sgn(cd % (a - c)), t2 = sgn(cd % (b - c));
    return !(r1 * r2 > 0 or r1 + r2 == -1 or t1 * t2 > 0);
    // 真相交或者 c-d 贴着 a-b 左边
}
std::string inPoly(const pii &a, pii *p, int n) {
    // 判断 a 点在多边形 p 的内/上/外
    int res = 0;
    pii b = {int(1e9 + 1), a.y};
    for (int i = 0; i < n; i++) { // 这里要有 p[n] = p[0]
        if (onSeg(a, p[i], p[i + 1])) return "ON";
        res += isCross(a, b, p[i], p[i + 1]);
    }
    return (res % 2 ? "IN" : "OUT");
}
```

7-7 点在不在凸包内

$O (\log n)$

```
bool onSeg(const pii &a, pii b, pii c) {
    // a on Seg b-c
    b = b - a, c = c - a;
    return (b % c == 0 and b * c <= 0);
}
std::string inConvex(const pii &a, pii *p, int n) {
    // 判断 a 点在凸包 p 的内/上/外
    if (a == p[0])
        return "ON";
    auto v = a - p[0];
    if ((p[1] - p[0]) % v < 0 or (p[n - 1] - p[0]) % v > 0)
        return "OUT";
    int l = 1, r = n - 1;
    while (l + 1 < r) {
        int mid = l + r >> 1;
        i64 cs = (p[mid] - p[0]) % v;
        if (cs == 0)
            return (p[mid] == a ? "ON" : (onSeg(a, p[0], p[mid]) ? "IN" : "OUT"));
        (cs > 0 ? l : r) = mid;
    }
    i64 cs = (p[r] - p[l]) % (a - p[l]);
    if (cs == 0 or onSeg(a, p[0], p[l]) or onSeg(a, p[0], p[r]))
        return "ON";
    return (cs > 0 ? "IN" : "OUT");
}
```

7-8 直线和凸包关系

$O(n)$ 预处理 $O(\log n)$ 查询

```cpp
const ld pi = acosl(-1);
ld fix(const ld &a) {
    if (a >= 1.5 * pi)
        return a - 2 * pi;
    return (a < -0.5 * pi ? a + 2 * pi : a);
}
int sgn(const i64 &a) {
    return (a < 0 ? -1 : a > 0);
}
const int N = 1e5 + 20;
std::string crossConvex(pii u, pii v, pii *p, int n) {
    // 判断直线 u-v 和凸包 p 的相交情况
    if (n == 0) return "OUT";
    if (n == 1)
        return ((u - v) % (p[0] - v) == 0 ? "ON" : "OUT");
    static ld ag[N]; // 把 N 写在函数前
    static int Capps = [&] {
        // 预处理边的倾角, 多个凸包的话不能这么写
        for (int i = 0; i < n; i++) {
            auto w = p[i + 1] - p[i];
            ag[i] = fix(atan2l(w.y, w.x));
        }
        return 0;
    }();
    auto b = u - v;
    ld th = fix(atan2l(b.y, b.x));
    int i = std::lower_bound(ag, ag + n, th) - ag;
    th = fix(th + pi);
    int j = std::lower_bound(ag, ag + n, th) - ag;
    int r1 = sgn(b % (p[i] - v)), r2 = sgn(b % (p[j] - v));
    return r1 * r2 > 0 ? "OUT" : (r1 * r2 < 0 ? "IN" : "ON");
}
```

7-9 直线切多边形

```cpp
void cutPoly(pii *p, int &n, pii a, pii b) {
    // 把多边形 p 在有向直线 a->b 左边的部分切开留下
    std::vector<pii> v;
    auto c = b - a;
    for (int i = 0; i < n; i++) {
        auto cr1 = c % (p[i] - a), cr2 = c % (p[(i + 1) % n] - a);
        if (sgn(cr1) >= 0)
            v.push_back(p[i]);
        if (sgn(cr1) * sgn(cr2) == -1)
            v.push_back(cross(a, b, p[i], p[(i + 1) % n]));
    }
    n = 0;
    for (auto po : v)
        p[n++] = po;
}
```

7-10 闵可夫斯基和

```
void Minkowski(pii *a, int n, pii *b, int m, pii *c, int &s) {
    // 闵可夫斯基和求两凸包的和, 以长度为 s 的 c 数组返回
    c[0] = a[0] + b[0], s = 1;
    int i = 0, j = 0;
    while (i < n and j < m)
        if ((a[(i + 1) % n] - a[i]) % (b[(j + 1) % m] - b[j]) >= 0)
            c[s] = c[s - 1] + a[(i + 1) % n] - a[i], s++, i++;
        else
            c[s] = c[s - 1] + b[(j + 1) % m] - b[j], s++, j++;
    while (i < n)
        c[s] = c[s - 1] + a[(i + 1) % n] - a[i], s++, i++; // copy
    while (j < m)
        c[s] = c[s - 1] + b[(j + 1) % m] - b[j], s++, j++; // copy
    convex(c, s);
}
```

7-11 夹角和四点共圆

```
#define pll std::pair<i64, i64>
pll angle(const pii &a, const pii &b, const pii &c) {
    // a-b 和 a-c 的夹角 tan
    auto v1 = b - a, v2 = c - a;
    if (v1 * v2 == 0)
        return {0, 1};             // 表示v1和v2垂直
    return {(v1 % v2), (v1 * v2)}; // tan∠bac
}
int conCircle(const pii &a, const pii &b, const pii &c, const pii &d) {
    // 对四个不同的点判断四点共圆
    if ((a - b) % (a - c) == 0 or (d - c) % (d - b) == 0)
        return 0;
    auto a1 = angle(a, b, c), a2 = angle(d, c, b);
    return ((__int128)a1.x * a2.y + (__int128)a1.y * a2.x) == 0;
}
```

7-12 平面最近点对

```
ld Closest(pii *p, int n) {
    std::sort(p, p + n); // 把点按 x 排序
    ld ans = 1e10;       // 整点则ans存dis2
    std::function<void(int, int)> find = [&](int l, int r) {
        if (r <= l)  return;
```

```
            int mid = l + r >> 1;
            find(l, mid); // 分治下去
            find(mid + 1, r);
            std::vector<pii> b; // 存中间缝隙的点
            for (int i = l; i <= r; i++)
                if (std::fabs(p[i].x - p[mid].x) < ans) // 整点则换成pf(Δx)
                    b.push_back(p[i]);
            std::sort(b.begin(), b.end(), [&](pii u, pii v) {
                return u.y < v.y; // 按 y 排序
            });
            for (int i = 0; i < b.size(); i++)
                for (int j = i + 1; j < b.size() and b[j].y - b[i].y < ans; j++)
                    ans = std::min(ans, dis(b[j] - b[i])); // 整点则条件变成pf(Δy)<ans
            // 答案只可能在这个夹缝里，更新答案
        };
        find(0, n - 1); // 记得调用函数
        return ans;
    }
    // O(nlog2n) 但几乎O(nlogn)
```

7-13 最小圆覆盖

```
ld dis(const pii &a, const pii &b) {
    auto p = a - b;
    return sqrtl(p.x * p.x + p.y * p.y);
}
pii cross(const pii &a, const pii &b, const pii &c, const pii &d) {
    // line a-b 交 line c-d
    auto v = c - d;
    ld sa = v % (a - d), sb = (b - d) % v;
    ld rate = sa / (sa + sb);
    return rate * (b - a) + a;
}
pii rot(const pii &a) {
    // 逆时针旋转90度
    return {-a.y, a.x};
}
pii rot(const pii &a, const ld &th) {
    // 逆时针旋转 th rad
    // 复数乘法 (a.x+a.yi)*(b.x+b.yi)=a.x*b.x-a.y*b.y+(a.x*b.y+a.y*b.x)i
    pii b = {cosl(th), sinl(th)};
    return {a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x};
}
pii excir(const pii &a, const pii &b, const pii &c) {
    // 三点求外接圆 圆心
    auto d1 = 0.5f * (a + b), d2 = d1 + rot(a - b);
    auto d3 = 0.5f * (c + b), d4 = d3 + rot(c - b);
    return cross(d1, d2, d3, d4);
}

pii p[N];
```

```
void minCirOver(int n, pii &Center, ld &R) {
    Center = {0, 0}, R = -1;
    // Random
    srand(time(0));
    std::random_shuffle(p + 1, p + n + 1);
    for (int i = 1; i <= n; i++) {
        if (dis(Center, p[i]) <= R)
            continue;
        // i点不在圆内,说明 i要更新圆
        Center = p[i], R = 0;
        for (int j = 1; j < i; j++) {
            if (dis(Center, p[j]) <= R)
                continue;
            // j不在圆内,说明 j要更新圆
            Center = 0.5f * (p[i] + p[j]), R = dis(Center, p[i]);
            for (int k = 1; k < j; k++) {
                if (dis(Center, p[k]) <= R)
                    continue;
                // k不在圆内,说明 k要更新圆
                Center = excir(p[i], p[j], p[k]), R = dis(Center, p[i]);
            }
        }
    }
}
```

7-14 最小矩形覆盖

```
    convex(p, n);
    // 注意要特判只有两个点的情况
    auto calc1 = [&](int i, int j, int k) { return (p[j] - p[i]) % (p[k] - p[i]);
};
    auto calc2 = [&](int i, int j, int k) { return (p[j] - p[i]) * (p[k] - p[i]);
};
    auto nt = [&](int i) { return (i + 1) % n; };
    int i = 0, j = 1, l = 1, r = 0;
    /*
       o------j
      /        \
     l          r
      \        /
       i-----o
    */
    for (; i < n; i++) {
        while (sgn(calc1(i, nt(i), j) - calc1(i, nt(i), nt(j))) <= 0)
            j = nt(j);
        while (sgn(calc2(i, nt(i), r) - calc2(i, nt(i), nt(r))) <= 0)
            r = nt(r);
        if (i == 0)
            l = r;
        while (sgn(calc2(i, nt(i), l) - calc2(i, nt(i), nt(l))) >= 0)
```

```
            l = nt(l);
        auto hor = (p[nt(i)] - p[i]);
        ld len = hor * (p[r] - p[l]) / dis(hor); // --
        auto ver = rot(hor);
        ld het = ver * (p[j] - p[i]) / dis(ver); // |
        ans = std::max(ans, std::fabs(len * het));
    }
```

7-15 圆并

$O(n^2 \log n)$

```
std::vector<pii> seg; // 一个圆被覆盖的区域
struct Circle {
    pii cen{0, 0};
    int r{0};
    bool inside(const Circle &c) { // 判断圆是否被c包含
        ld d = dis(cen, c.cen);
        return sgn(r + d - c.r) <= 0;
    }
    static ld fix(ld th) {
        if (th >= 2 * pi)
            return th - 2 * pi;
        return (th < 0 ? th + 2 * pi : th);
    }
    void cross(const Circle &c) { // 找到c圆把自己交了哪个区域
        ld d = dis(cen, c.cen);
        if (sgn(d - r - c.r) >= 0) // 没交集
            return;
        auto v = c.cen - cen;
        ld mid = atan2(v.y, v.x);
        ld th = acos((r * r + d * d - c.r * c.r) / (2 * r * d));
        ld L = fix(mid - th), R = fix(mid + th);
        if (L < R)
            seg.push_back({L, R});
        else
            seg.push_back({0, R}), seg.push_back({L, 2 * pi});
    }
    ld bow(const ld &a) { // 弓形面积
        return (a - sin(a)) * r * r;
    }
    ld edge(ld L, ld R) { // 边面积
        auto a = cen + (pii){r * cosl(L), r * sinl(L)};
        auto b = cen + (pii){r * cosl(R), r * sinl(R)};
        return (a % b);
    }
    ld calc() { // 计算没被别的圆交的区域的面积
        ld ans = 0;
        seg.push_back({0, 0});
        seg.push_back({2 * pi, 2 * pi});
        std::sort(seg.begin(), seg.end());
        ld lim = 0;
        for (auto [L, R] : seg)
            if (L > lim) {
                ans += bow(L - lim) + edge(lim, L);
                lim = R;
            } else
                lim = std::max(lim, R);
        seg.clear();
        return ans;
    }
};

ld CircleUnion(Circle *c, int n) { // 返回圆并面积
    for (int i = 0; i < n; i++) // 去除被包含的圆
        for (int j = 0; j < n; j++)
            if (i != j and c[i].inside(c[j])) {
                c[i--] = c[--n];
                break;
            }
    ld ans = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            if (i != j)
                c[i].cross(c[j]); // 看看j圆把i圆切了哪儿了
        ans += c[i].calc();  // 计算i圆的贡献
    }
    return ans / 2; // 之前计算的都是两倍面积
}
```

7-16 三维计算几何

```
node operator%(const node &a, const node &b) {
    // 叉乘
    node ans;
    ans.x = a.y * b.z - b.y * a.z;
    ans.y = b.x * a.z - a.x * b.z;
    ans.z = a.x * b.y - b.x * a.y;
    return ans;
}
int operator*(const node &a, const node &b) {
    // 点乘
    return a.x * b.x + a.y * b.y + a.z * b.z;
}
node operator*(int a, node b) {
    // 数乘
    b.x *= a;
    b.y *= a;
    b.z *= a;
    return b;
}
bool sameLine(node a, node b, node c) {
    // 判断三点共线
    b = b - a;
    c = c - a;
    return b % c == zero;
}
bool samePlane(node a, node b, node c, node d) {
    // 判断三点共面
    a = a - d;
    b = b - d;
    c = c - d;
    return (a % b) * c == 0;
}
```

7-17 三维四点共圆

```
struct angle {
    int dot{0};
    node cross{0, 0, 0};
};
angle get(node a, node b, node c) {
    // 求∠bac
    b = b - a;
    c = c - a;
    angle ans;
    ans.dot = b * c;
    ans.cross = b % c;
    return ans;
}
bool sameCircle(node a, node b, node c, node d) {
    // 基于四点共面判断四点是否共圆
    // 其中不能出现三点共线
    auto ag1 = get(a, b, c);
    auto ag2 = get(d, c, b);
    return ag1.dot * ag2.cross + ag2.dot * ag1.cross == zero;
    //注意仔细算算会不会爆longlong
}
// 也可以在基于四点共面&没有三点共线的前提下用托勒密定理判
```