- 动态规划
  - 背包
    - 01 背包
    - 完全背包
    - 二进制优化多重背包
    - 单调队列优化多重背包
    - 混合背包
    - 背包求具体方案(字典序最小)
  - 数位DP
    - 区间数位和
    - 区间数位个数
    - 相邻数字有限制
  - 子序列DP
    - 最长上升子序列
    - 最长公共上升子序列
  - SOSDP
- 计算几何
  - 基础知识
    - 浮点比较
    - 常见结构封装
    - 常见函数
  - 凸包
    - 凸包
    - 上下凸包
    - 判断点是否在凸包内
    - 求平行于给定直线的凸多边形的切线
    - 判断直线与凸包是否相交
    - 过凸多边形外一点求凸多边形的切线
    - 闵可夫斯基和 判断凸包和凸包的关系
    - 凸包新加 k 个点后新凸包的面积
    - 动态凸包
  - 旋转卡壳
    - 平面最远点对
    - 最小矩形覆盖
  - 半平面交
    - 排序增量法，复杂度 O(nlogn)
    - 凸多边形内的最大内切圆半径
  - 面积并
    - 多边形面积并
    - 矩形面积并 O(nlogn)
  - 面积最大、最小三角形
    - 面积最大三角形
    - 极角序扫描线，复杂度 O(n^2logn)
  - 判断多个线段是否存在交点
  - 平面最近点对
  - 三维凸包
  - 线段与多边形公共部分的长度
  - 圆
    - 三角形外接圆
    - 三角形内接圆
    - 通过一点作圆的切线 返回角度
    - 通过一点并切于一条直线的圆
    - 与两条直线相切的圆 返回圆心
    - 与两个不相交的圆外切的圆 返回圆心
    - 圆与多边形面积交 记得输出绝对值
    - 圆面积并 记得输出绝对值
    - 最小圆覆盖
  - 自适应辛普森积分
- 树、图
  - 2-SAT
  - 差分约束
    - 最长路
    - 最短路
  - 二分图
    - 染色法判断二分图
    - 匈牙利算法求二分图最大匹配 O(nm)
  - 连通性问题
    - 有向图的强连通分量
    - 最大半连通子图就是缩点后的最长链
    - 边双连通分量
    - 点双连通分量
    - 圆方树
  - 欧拉回路和路径
    - 欧拉回路
    - 欧拉路径
  - 全局最小割
  - 树上K级祖先
  - 斯坦纳树
  - 稳定婚姻系统
  - 无向图三元环计数

- 虚树
- 严格次小生成树
- 一般图匹配
  - 一般图最大匹配 O(n^3)
  - 一般图最大权匹配 O(n^3)
- 最短哈密顿路径
- 最短路和次短路及方案数
  - 正权图
  - 无向无权图
- 最小树形图
- bellman_ford
- Boruvka
- djs
  - 朴素 djs
  - 堆优化的 djs
- floyd
  - 多源最短路
  - 传递闭包
  - 无向图最小环
  - 从起点到终点恰好经过 k 条边的最短路（可重复经过边）
- Johnson
- K短路
- Kruskal
  - 最小生成树
  - Kruskal 重构树
- LCA
- Prim
  - 朴素 Prim O(n^2) 适合稠密图
  - 堆优化 prim 不如 kruskal
- prufer
- SPFA
  - SPFA 求最短路
  - 判断负环

数据结构

- 并查集
- 单调栈
- 单调队列
  - 找出滑动窗口中的最大值/最小值
  - 求 k * k 的矩阵中的最大值和最小值
- 笛卡尔树
- 珂朵莉树
- 科技
- 可持久化数据结构
  - 可持久化Trie
  - 主席树
- 李超线段树
- 莫队
  - 普通莫队
  - 带修莫队
  - 回滚莫队
  - 树上莫队
  - 二次离线莫队
- 树链剖分
- 树套树
  - 支持单点修改，查询一个区间内的前驱
- 树状数组
- 线段树
  - 求区间最大连续子段和
  - 区间最大公约数 支持区间加，区间查询
  - 区间加，区间求和
  - 区间加, 区间极值
  - 区间加，区间乘，区间求和
- 压缩Trie树
- 左偏树、可并堆
- CDQ分治
  - 三维偏序
  - 二维区域和、二维数点
  - 动态逆序对
- DSU on tree
- LCT
- Splay
  - 区间翻转
- ST表
- Treap
- Trie
  - 最大异或对

数学

- 本源勾股数
- 多项式
  - NTT
  - 分治NTT

动态规划

背包

01 背包

```
int n, m;
int v[N], w[N];
int f[N];
void solve() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> v[i] >> w[i];
    for (int i = 1; i <= n; i++) {
        for (int j = m; j >= v[i]; j--) {
            f[j] = max(f[j], f[j - v[i]] + w[i]);
        }
    }
    cout << f[m] << '\n';
}
```

完全背包

```
int n, m;
int v[N], w[N];
int f[N];
void solve() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> v[i] >> w[i];
    for (int i = 1; i <= n; i++) {
        for (int j = v[i]; j <= m; j++) {
            f[j] = max(f[j], f[j - v[i]] + w[i]);
        }
    }
    cout << f[m];
}
```

二进制优化多重背包

```
int v[N], w[N], cnt;
int f[N];
void solve() {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        int a, b, s;
        cin >> a >> b >> s;
        int k = 1;
        while (k <= s) {
            cnt++;
            v[cnt]= a * k;
            w[cnt]= b * k;
            s -= k;
            k *= 2;
        }
        if (s > 0) {
            cnt++;
            v[cnt] = a * s;
            w[cnt] = b * s;
        }
    }
    n = cnt;
    for (int i = 1; i <= n; i++) {
        for (int j = m; j >= v[i]; j--) {
            f[j] = max(f[j], f[j - v[i]] + w[i]);
        }
    }
    cout << f[m];
}
```

单调队列优化多重背包

```
int f[N], q[N], g[N];
void solve() {
    int n, m;
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        int v, w, s;
        cin >> v >> w >> s;
        memcpy(g, f, sizeof(f));
        for (int j = 0; j < v; j++) {
            int hh = 0, tt = -1;
            for (int k = j; k <= m; k += v) {
                if (hh <= tt && (q[hh] < k - s * v)) hh++;
                while (hh <= tt && (g[q[tt]] - (q[tt] - j) / v * w) <= (g[k] - (k - j) / v * w)) tt--;
```

```
                q[++tt] = k;
                f[k] = g[q[hh]] + (k - q[hh]) / v * w;
            }
        }
    }
    cout << f[m];
}
```

混合背包

```
int f[1002], g[1002], q[1002];
void solve() {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        int v, w, s;
        cin >> v >> w >> s;
        if (s == -1) s = 1;                         // 01背包就是 s = 1 的多重背包
        else if (s == 0) s = m / v;                 // 完全背包就是 s = m / v 的多重背包
        memcpy(g, f, sizeof(f));
        for (int j = 0; j < v; j++) {
            int hh = 0, tt = -1;
            for (int k = j; k <= m; k += v) {
                while (hh <= tt && q[hh] < k - s * v) hh++;
                while (hh <= tt && g[q[tt]] - (q[tt] - j) / v *w <= g[k] - (k - j) / v * w) tt--;
                q[++tt] = k;
                f[k] = g[q[hh]] + (k - q[hh]) / v * w;
            }
        }
    }
    cout << f[m];
}
```

背包求具体方案(字典序最小)

```
int v[1002], w[1002], f[1002][1002];
void solve() {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        cin >> v[i] >> w[i];
    }
    for (int i = n; i >= 1; i--) {                                                  // 倒着处理方便正着输出
        for (int j = 0; j <= m; j++) {
            f[i][j] = f[i + 1][j];
            if (j >= v[i]) f[i][j] = max(f[i][j], f[i + 1][j - v[i]] + w[i]);
        }
    }
    int k = m;
    for (int i = 1; i <= n; i++) {
        if (k >= v[i] && f[i][k] == f[i + 1][k - v[i]] + w[i]) {                 // 可以选就必选
            cout << i << ' ';
            k -= v[i];
        }
    }
}
```

数位DP

区间数位和

```
using LL = long long;
const int mod = 1e9 + 7;
LL dp[20][200]; // 在前 pos 个位置，数位和为 sum 的情况下的总和
int a[20];
LL dfs(int pos, bool limit, int sum) {
    if (!pos) return sum;
    if (!limit && ~dp[pos][sum]) return dp[pos][sum];
    int up = limit ? a[pos] : 9;
    LL res = 0;
    for (int i = 0; i <= up; i++) res = (res + dfs(pos - 1, limit && i == up, sum + i)) % mod;
    if (!limit) dp[pos][sum] = res;
    return res;
}
LL get(LL x) {
    int len = 0;
    while (x) {
        a[++len] = x % 10;
        x /= 10;
```

```
    }
    return dfs(len, 1, 0);
}
void solve() {
    LL l, r;
    cin >> l >> r;
    LL ans = 0;
    ans = (ans + get(r)) % mod;
    ans = (ans - get(l - 1)) % mod;
    ans = (ans % mod + mod) % mod;
    cout << ans << '\n';
}
signed main() {
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    memset(dp, -1, sizeof(dp));
    int T = 1;
    cin >> T;
    while (T--) solve();
    return 0;
}
```

区间数位个数

```
// f[pos][cnt]：当最高位在 [pos + 1, len] 中 digit 填了 cnt 个，[1, pos] 任意填，digit 出现的次数
// 当 limit = false 时，容易知道填 1-9 的数量是相同的
using LL = long long;
LL dp[20][20], ans[10];
int a[20];
int digit; // 当前要统计的数字
LL dfs(int pos, bool limit, bool lead0, int cnt) {
    if (!pos) return cnt;
    if (!limit && ~dp[pos][cnt] && !lead0) return dp[pos][cnt];
    int up = limit ? a[pos] : 9;
    LL res = 0;
    for (int i = 0; i <= up; i++) {
        int tmp = cnt + (i == digit);
        if (lead0 && digit == 0 && i == 0) tmp = 0;
        res += dfs(pos - 1, limit && i == up, lead0 && i == 0, tmp);
    }
    if (!limit && !lead0) dp[pos][cnt] = res;
    return res;
}
void get(LL x, int type) {
    int len = 0;
    while (x) {
        a[++len] = x % 10;
        x /= 10;
    }
    for (int i = 0; i <= 9; i++) {
        digit = i;
        ans[i] += type * dfs(len, 1, 1, 0);
    }
}
void solve() {
    memset(dp, -1, sizeof(dp));
    LL l, r;
    cin >> l >> r;
    get(r, 1);
    get(l - 1, -1);
    for (int i = 0; i <= 9; i++) cout << ans[i] << ' ';
}
```

相邻数字有限制

```
// 不含前导零且相邻两个数字之差至少为 2 的正整数被称为windy数
// f[pos][last]：长度为 pos + 1 的以 last 开头的 windy 数的数量
// 相当于 [1, pos] 没有填，第 pos + 1 位填了 last
using LL = long long;
LL dp[20][10];
int a[20];
LL dfs(int pos, bool limit, bool lead0, int last) {
    if (!pos) return 1;
    if (!limit && last >= 0 && last <= 9 && ~dp[pos][last]) return dp[pos][last];
    int up = limit ? a[pos] : 9;
    LL res = 0;
    for (int i = 0; i <= up; i++) {
        if (lead0) res += dfs(pos - 1, limit && i == up, lead0 && i == 0, i == 0 ? last : i);
        else if (abs(last - i) >= 2) res += dfs(pos - 1, limit && i == up, lead0 && i == 0, i);
    }
```

```
    if (!limit && last >= 0 && last <= 9) dp[pos][last] = res;
    return res;
}
LL get(LL x) {
    int len = 0;
    while (x) {
        a[++len] = x % 10;
        x /= 10;
    }
    return dfs(len, 1, 1, -5);
}
void solve() {
    memset(dp, -1, sizeof(dp));
    LL l, r;
    cin >> l >> r;
    LL ans = 0;
    ans += get(r);
    ans -= get(l - 1);
    cout << ans << '\n';
}
```

子序列DP

最长上升子序列

```
// 贪心 + 二分 存路径相同下的最小值
int a[N];
int q[N];
void solve() {
    q[0] = -2e9;
    int n;
    cin >> n;
    int len = 0;
    for (int i = 0; i < n; i++) cin >> a[i];
    for (int i = 0; i < n; i++) {
        int l = 0, r = len;
        while (l < r) {
            int mid = (l + r + 1) >> 1;
            if (q[mid] < a[i]) l = mid;
            else r = mid - 1;
        }
        len = max(len, r + 1);
        q[r + 1] = a[i];
    }
    cout << len;
}
```

最长公共上升子序列

```
int a[3002], b[3002];
int f[3002][3002];
void solve() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    for (int j = 1; j <= n; j++) cin >> b[j];
    for (int i = 1; i <= n; i++) {
        int maxv = 1;
        for (int j = 1; j <= n; j++) {
            f[i][j] = f[i - 1][j];
            if (a[i] == b[j]) f[i][j] = max(f[i][j], maxv);
            if (a[i] > b[j]) maxv = max(maxv, f[i - 1][j] + 1);
        }
    }
    int res = 0;
    for (int i = 1; i <= n; i++) res = max(f[n][i], res);
    cout << res;
}
```

SOSDP

```
for (int i = 0; i < n; i++) {
    for (int st = 0; st < (1 << n); st++) {
        if (st >> i & 1) {
            // dp[st] = max(dp[st], dp[st ^ (1 << i)]); 集合值为所有子集的最大值
            // dp[st] += dp[st ^ (1 << i)]; 集合值为所有子集和
        }
    }
}
```

计算几何

基础知识

能用整数就不要使用浮点

求交点要尽量先判断（可行性），再求交

算极角 atan2l(y, x)

皮克定理是指一个计算点阵中顶点在格点上的多边形面积公式该公式可以表示为:

S = a + b / 2 - 1

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

海伦公式 p = (a + b + c) / 2;

S = sqrtl(p(p - a) * (p - b) * (p - c));

三角形四心

(1) 外心，外接圆圆心

三边中垂线交点。到三角形三个顶点的距离相等

(2) 内心，内切圆圆心

角平分线交点，到三边距离相等

(3) 垂心

三条垂线交点

(4) 重心

三条中线交点（到三角形三顶点距离的平方和最小的点，三角形内到三边距离之积最大的点）

浮点比较

```
const long double eps = 1e-8, PI = acosl(-1);
int sign(long double x) { // 符号函数
    if (abs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}
int cmp(long double x, long double y) { // 比较函数
    if (abs(x - y) < eps) return 0;
    if (x < y) return -1;
    return 1;
}
```

常见结构封装

```
struct Point {
    long double x, y;
    int id = -1;
    Point(long double x = 0, long double y = 0) : x(x), y(y) {}
    bool operator ==(const Point &b) const {
        return abs(x - b.x) <= eps && abs(y - b.y) <= eps;
    }
    bool operator <(const Point &b) const {
        if (abs(x - b.x) <= eps) return y < b.y - eps;
        return x < b.x - eps;
    }
    bool operator >(const Point &b) const {
```

```
        return !(*this < b || *this == 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);
    }
    Point operator -() const { return {-x, -y}; }
    Point operator *(const long double &t) const {
        return {x * t, y * t};
    }
    Point operator /(const long double &t) const {
        return {x / t, y / t};
    }
    long double operator *(const Point &b) const {
        return x * b.x + y * b.y;
    }
    // 叉积 向量 A 与 B 张成的平行四边形的有向面积
    // P ^ Q > 0, P 在 Q 的顺时针方向; < 0, P 在 Q 的逆时针方向; = 0, P, Q 共线, 可能同向或反向
    long double operator ^(const Point &b) const {
        return x * b.y - b.x * y;
    }
    int toleft(const Point &b) const {
        const auto t = (*this) ^ b;
        return (t > eps) - (t < -eps);
    }
    // 浮点数
    long double len() const { // 向量长度
        return sqrtl(x * x + y * y);
    }
    long double dis(const Point &a) const { // 两点距离
        return sqrtl((a.x - x) * (a.x - x) + (a.y - y) * (a.y - y));
    }
    long double ang(const Point &a) const { // 向量夹角
        return abs(atan2l((*this) ^ a, (*this) * a));
        // return acosl(max(-1.0l, min(1.0l, ((*this) * a) / (len() * a.len()))));
    }
    Point rot(const long double rad) const {  // 逆时针旋转（给定角度）
        return {x * cosl(rad) - y * sinl(rad), x * sinl(rad) + y * cosl(rad)};
    }
    Point rot(const long double cosr, const long double sinr) const { // 逆时针旋转（给定角度的正弦与余弦）
        return {x * cosr - y * sinr, x * sinr + y * cosr};
    }
};
// 极角排序
struct argcmp {
    bool operator()(const Point &a, const Point &b) const {
        const auto quad = [](const Point &a) {
            if (a.y < -eps) return 1;
            if (a.y > eps) return 4;
            if (a.x < -eps) return 5;
            if (a.x > eps) return 3;
            return 2;
        };
        const int qa = quad(a), qb = quad(b);
        if (qa != qb) return qa < qb;
        const auto t = a ^ b;
        if (abs(t) <= eps) return a * a < b * b - eps;  // 不同长度的向量需要分开
        return t > eps;
    }
};
struct Line {
    Point p, v;
    bool operator ==(const Line &b) const {
        return v.toleft(b.v) == 0 && v.toleft(p - b.p) == 0;
    }
    int toleft(const Point &b) const {
        return v.toleft(b - p);
    }
    bool operator <(const Line &b) const { // 半平面交算法定义的排序
        if (abs(v ^ b.v) <= eps && v * b.v >= -eps) return toleft(b.p) == -1;
        return argcmp()(v, b.v);
    }
    // 浮点数
    Point inter(const Line &a) const { // 直线交点
        return p + v * ((a.v ^ (p - a.p)) / (v ^ a.v));
    }
    long double dis(const Point &a) const { // 点到直线距离
        return abs(v ^ (a - p)) / v.len();
    }
```

```
    Point proj(const Point &a) const { // 点在直线上的投影
        return p + v * ((v * (a - p)) / (v * v));
    }
};
struct Segment {
    Point a, b;
    bool operator<(const Segment &s) const {return make_pair(a, b) < make_pair(s.a, s.b);}
    // 判定性函数建议在整数域使用
    // 判断点是否在线段上
    // -1 点在线段端点 | 0 点不在线段上 | 1 点严格在线段上
    int is_on(const Point &p) const {
        if (p == a || p == b) return -1;
        return (p - a).toleft(p - b) == 0 && (p - a) * (p - b) < -eps;
    }
    // 判断线段直线是否相交
    // -1 直线经过线段端点 | 0 线段和直线不相交 | 1 线段和直线严格相交
    int is_inter(const Line &l) const {
        if (l.toleft(a) == 0 || l.toleft(b) == 0) return -1;
        return l.toleft(a) != l.toleft(b);
    }
    // 判断两线段是否相交
    // -1 在某一线段端点处相交 | 0 两线段不相交 | 1 两线段严格相交
    int is_inter(const Segment &s) const {
        if (is_on(s.a) || is_on(s.b) || s.is_on(a) || s.is_on(b)) return -1;
        const Line l{a, b - a}, ls{s.a, s.b - s.a};
        return l.toleft(s.a) * l.toleft(s.b) == -1 && ls.toleft(a) * ls.toleft(b) == -1;
    }
    // 点到线段距离
    long double dis(const Point &p) const {
        if ((p - a) * (b - a) < -eps || (p - b) * (a - b) < -eps) return min(p.dis(a), p.dis(b));
        const Line l{a, b - a};
        return l.dis(p);
    }
    // 两线段间距离
    long double dis(const Segment &s) const {
        if (is_inter(s)) return 0;
        return min({dis(s.a), dis(s.b), s.dis(a), s.dis(b)});
    }
};
struct Circle {
    Point c;
    long double r;
    bool operator ==(const Circle &a) const {
        return c == a.c && abs(r - a.r) <= eps;
    }
    long double circ() const {return 2 * PI * r;}  // 周长
    long double area() const {return PI * r * r;}  // 面积
    // 点与圆的关系
    // -1 圆上 | 0 圆外 | 1 圆内
    int is_in(const Point &p) const {
        const long double d = p.dis(c);
        return abs(d - r) <= eps ? -1 : d < r - eps;
    }
    // 直线与圆关系
    // 0 相离 | 1 相切 | 2 相交
    int relation(const Line &l) const {
        const long double d = l.dis(c);
        if (d > r + eps) return 0;
        if (abs(d - r) <= eps) return 1;
        return 2;
    }
    // 圆与圆关系
    // -1 相同 | 0 相离 | 1 外切 | 2 相交 | 3 内切 | 4 内含
    int relation(const Circle &a) const {
        if (*this == a) return -1;
        const long double d = c.dis(a.c);
        if (d > r + a.r + eps) return 0;
        if (abs(d - r - a.r) <= eps) return 1;
        if (abs(d - abs(r - a.r)) <= eps) return 3;
        if (d < abs(r - a.r) - eps) return 4;
        return 2;
    }
    // 直线与圆的交点
    vector<Point> inter(const Line &l) const {
        const long double d = l.dis(c);
        const Point p = l.proj(c);
        const int t = relation(l);
        if (t == 0) return vector<Point>();
        if (t == 1) return vector<Point>{p};
        const long double k = sqrtl(r * r - d * d);
        return vector<Point>{p - (l.v / l.v.len()) * k, p + (l.v / l.v.len()) * k};
```

```
    }
    // 圆与圆交点
    vector<Point> inter(const Circle &a) const {
        const long double d = c.dis(a.c);
        const int t = relation(a);
        if (t == -1 || t == 0 || t == 4) return vector<Point>();
        Point e = a.c - c; e = e / e.len() * r;
        if (t == 1 || t == 3) {
            if (r * r + d * d - a.r * a.r >= -eps) return vector<Point>{c + e};
            return vector<Point>{c - e};
        }
        const long double costh = (r * r + d * d - a.r * a.r) / (2 * r * d), sinth = sqrtl(1 - costh * costh);
        return vector<Point>{c + e.rot(costh, -sinth), c + e.rot(costh, sinth)};
    }
    // 圆与圆交面积
    long double inter_area(const Circle &a) const {
        const long double d = c.dis(a.c);
        const int t = relation(a);
        if (t == -1) return area();
        if (t < 2) return 0;
        if (t > 2) return min(area(), a.area());
        const long double costh1 = (r * r + d * d - a.r * a.r) / (2 * r * d), costh2 = (a.r * a.r + d * d - r * r) / (2
* a.r * d);
        const long double sinth1 = sqrtl(1 - costh1 * costh1), sinth2 = sqrtl(1 - costh2 * costh2);
        const long double th1 = acosl(costh1), th2 = acosl(costh2);
        return r * r * (th1 - costh1 * sinth1) + a.r * a.r * (th2 - costh2 * sinth2);
    }
    // 过圆外一点圆的切线
    vector<Line> tangent(const Point &a) const {
        const int t = is_in(a);
        if (t == 1) return vector<Line>();
        if (t == -1) {
            const Point v = {-(a - c).y, (a - c).x};
            return vector<Line>{{a, v}};
        }
        Point e = a - c; e = e / e.len() * r;
        const long double costh = r / c.dis(a), sinth = sqrtl(1 - costh * costh);
        const Point t1 = c + e.rot(costh, -sinth), t2 = c + e.rot(costh, sinth);
        return vector<Line>{{a, t1 - a}, {a, t2 - a}};
    }
    // 两圆的公切线
    vector<Line> tangent(const Circle &a) const {
        const int t = relation(a);
        vector<Line> lines;
        if (t == -1 || t == 4) return lines;
        if (t == 1 || t == 3) {
            const Point p = inter(a)[0], v = {-(a.c - c).y, (a.c - c).x};
            lines.push_back({p, v});
        }
        const long double d = c.dis(a.c);
        const Point e = (a.c - c) / (a.c - c).len();
        if (t <= 2) {
            const long double costh = (r - a.r) / d, sinth = sqrtl(1 - costh * costh);
            const Point d1 = e.rot(costh, -sinth), d2 = e.rot(costh, sinth);
            const Point u1 = c + d1 * r, u2 = c + d2 * r, v1 = a.c + d1 * a.r, v2 = a.c + d2 * a.r;
            lines.push_back({u1, v1 - u1}); lines.push_back({u2, v2 - u2});
        }
        if (t == 0) {
            const long double costh = (r + a.r) / d, sinth = sqrtl(1 - costh * costh);
            const Point d1 = e.rot(costh, -sinth), d2 = e.rot(costh, sinth);
            const Point u1 = c + d1 * r, u2 = c + d2 * r, v1 = a.c - d1 * a.r, v2 = a.c - d2 * a.r;
            lines.push_back({u1, v1 - u1}); lines.push_back({u2, v2 - u2});
        }
        return lines;
    }
    // 圆外的点的反演点在圆内，反之亦然；圆上的点的反演点为其自身。
    // 不过圆心 O 的圆其反演图形也是不过圆心 O 的圆
    // 过点 O 的圆，其反演图形是不过点 O 的直线
    // 两个图形相切且存在不为点 O 的切点，则他们的反演图形也相切。
    // 圆的反演
    tuple<int, Circle, Line> inverse(const Line &l) const {
        const Circle null_c = {{0.0, 0.0}, 0.0};
        const Line null_l = {{0.0, 0.0}, {0.0, 0.0}};
        if (l.toleft(c) == 0) return {2, null_c, l};
        const Point v = l.toleft(c) == 1 ? Point{l.v.y, -l.v.x} : Point{-l.v.y, l.v.x};
        const long double d = r * r / l.dis(c);
        const Point p = c + v / v.len() * d;
        return {1, {(c + p) / 2, d / 2}, null_l};
    }
    tuple<int, Circle, Line> inverse(const Circle &a) const {
        const Circle null_c = {{0.0, 0.0}, 0.0};
```

```
        const Line null_l = {{0.0, 0.0}, {0.0, 0.0}};
        const Point v = a.c - c;
        if (a.is_in(c) == -1) {
            const long double d = r * r / (a.r + a.r);
            const Point p = c + v / v.len() * d;
            return {2, null_c, {p, {-v.y, v.x}}};
        }
        if (c == a.c) return {1, {c, r * r / a.r}, null_l};
        const long double d1 = r * r / (c.dis(a.c) - a.r), d2 = r * r / (c.dis(a.c) + a.r);
        const Point p = c + v / v.len() * d1, q = c + v / v.len() * d2;
        return {1, {(p + q) / 2, p.dis(q) / 2}, null_l};
    }
};
```

常见函数

计算向量模长

```
long double get_length(Point a) {
    return sqrtl(a * a);
}
```

计算向量夹角

atan2l 返回的是一个有符号的角度 [-π, π]。

acosl 返回的是无符号角度 [0, π]

```
long double get_angle(Point a, Point b) {
    return atan2l(a ^ b, a * b);
}
// 同计算夹角，但是需要将输入限制在 [-1,1]
long double get_angle(Point a, Point b) {
    long double res = a * b / get_length(a) / get_length(b);
    res = min(1.0l, max(-1.0l, res));
    return acosl(res);
}
```

计算两向量 ab, ac 构成的平行四边形有向面积

```
long double area(Point a, Point b, Point c) {
    return (b - a) ^ (c - a);
}
```

向量顺时针选转后的向量 如果能直接知道 cos 和 sin，那么最好不要算 angle

```
Point rotate(Point a, long double angle) {
    return Point(a.x * cosl(angle) + a.y * sinl(angle), -a.x * sinl(angle) + a.y * cosl(angle));
}
```

两直线相交 v w 分别是直线的方向向量

```
using Vector = Point;
Point get_line_intersection(Point p, Vector v, Point q, Vector w) {
    Vector u = p - q;
    long double t = (w ^ u) / (v ^ w);
    return p + v * t;
}
```

点到直线的距离，a b 是直线两点

```
long double distance_to_line(Point p, Point a, Point b) {
    Vector v1 = b - a, v2 = p - a;
    return abs((v1 ^ v2) / get_length(v1));
}
```

点到线段距离

```
long double distance_to_segment(Point p, Point a, Point b) {
    if (a == b) return get_length(p - a);
    Vector v1 = b - a, v2 = p - a, v3 = p - b;
    if (sign(v1 * v2) < 0) return get_length(v2);
    if (sign(v1 * v3) > 0) return get_length(v3);
    return distance_to_line(p, a, b);
}
```

点在直线上的投影，a b 是直线两点

```
Point get_line_projection(Point p, Point a, Point b) {
    Vector v = b - a;
    return a + v * ((v * (p - a)) / (v * v));
}
```

判断点是否在线段上

```
bool on_segment(Point p, Point a, Point b) {
    return sign((p - a) ^ (p - b)) == 0 && sign((p - a) * (p - b)) <= 0;
}
```

判断两线段是否相交

```
bool segment_intersection(Point a1, Point a2, Point b1, Point b2) {
    if (min(a1.x, a2.x) > max(b1.x, b2.x) ||
        min(b1.x, b2.x) > max(a1.x, a2.x) ||
        min(a1.y, a2.y) > max(b1.y, b2.y) ||
        min(b1.y, b2.y) > max(a1.y, a2.y)) return 0;
    long double c1 = (a2 - a1) ^ (b1 - a1), c2 = (a2 - a1) ^ (b2 - a1);
    long double c3 = (b2 - b1) ^ (a2 - b1), c4 = (b2 - b1) ^ (a1 - b1);
    return sign(c1) * sign(c2) <= 0 && sign(c3) * sign(c4) <= 0;
}
```

求任意多边形面积, 点从 0 到 n - 1

```
long double polygon_area(Point p[], int n) {
    long double s = 0;
    for (int i = 1; i + 1 < n; i++) s += (p[i] - p[0]) ^ (p[i + 1] - p[i]);
    return s / 2;
}
```

极角排序 （直接在 sort 里比较时使用 atan2l 较慢，需预处理）

```
vector<pair<Point, long double>> a(n);
for (int i = 0; i < n; i++) {
    cin >> a[i].first.x >> a[i].first.y;
    a[i].second = atan2l(a[i].first.y, a[i].first.x);
}
sort(a.begin(), a.end(), [&](pair<Point, long double> a, pair<Point, long double> b) {
    return a.second < b.second;
});
```

判断点P在多边形内

```
bool InPolygon(Point P, vector<Point> &p) {
    bool flag = false;
    int n = p.size();
    Point P1, P2; // 多边形一条边的两个顶点
    for (int i = 0, j = n - 1; i < n; j = i++) {
        P1 = point[i];
        P2 = point[j];
        if (on_segment(P, P1, P2)) return true; // 点在多边形一条边上
        if ((sign(P1.y - P.y) > 0 != sign(P2.y - P.y) > 0) && sign(P.x - (P.y - P1.y) * (P1.x - P2.x) / (P1.y - P2.y) -
P1.x) < 0) flag = !flag;
    }
    return flag;
}
```

已知正方形对角求剩余两点

```
vector<Point> poly(4);
poly[0] = a, poly[2] = b;
poly[1] = {(a.x - a.y + b.x + b.y) / 2, (a.x + a.y - b.x + b.y) / 2};
poly[3] = {(a.x + a.y + b.x - b.y) / 2, (-a.x + a.y + b.x + b.y) / 2};
```

凸包

浮点数比较一定要带 sign

凸包 只有一条直线时会退化，可通过 andrew 后的 size 来判断,如果 size 等于 1 说明是一个直线,也可通过上下凸包解决

认真考虑共线

凸包

```
int sign(long double x) { // 符号函数
    if (abs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}
int cmp(long double x, long double y) {
    if (abs(x - y) < eps) return 0;
    else if (x < y) return -1;
    else return 1;
}
long double area(Point a, Point b, Point c) {
    return (b - a) ^ (c - a);
}
vector<Point> andrew(vector<Point> &q) {
    int n = q.size(), top = 0;
    vector<int> stk(n + 1);
    vector<bool> used(n + 1, 0);
    sort(q.begin(), q.end(), [&](Point a, Point b) {
        if (cmp(a.x, b.x) == 0) return a.y < b.y;
        else return a.x < b.x;
    });
    for (int i = 0; i < n; i++) {
        while (top >= 2 && sign(area(q[stk[top - 2]], q[stk[top - 1]], q[i])) <= 0) {  // < 0 可以把共线的都算上
            if (sign(area(q[stk[top - 2]], q[stk[top - 1]], q[i])) < 0) used[stk[--top]] = 0;
            else top--;
        }
        stk[top++] = i;
        used[i] = 1;
    }
    used[0] = 0;
    for (int i = n - 1; i >= 0; i--) {
        if (used[i]) continue;
        while (top >= 2 && sign(area(q[stk[top - 2]], q[stk[top - 1]], q[i])) <= 0) {
            if (sign(area(q[stk[top - 2]], q[stk[top - 1]], q[i])) < 0) used[stk[--top]] = 0;
            else top--;
        }
        stk[top++] = i;
        used[i] = 1;
    }
    top--; // 去掉最后一个点
    vector<Point> point;
    for (int i = 0; i < top; i++) point.push_back(q[stk[i]]);
    return point;
}
```

上下凸包

```
const long double eps = 1e-8;
long double area(Point a, Point b, Point c) {
    return (b - a) ^ (c - a);
}
int sign(long double x) { // 符号函数
    if (abs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}
int cmp(long double x, long double y) {
    if (abs(x - y) < eps) return 0;
    else if (x < y) return -1;
    else return 1;
}
pair<vector<Point>, vector<Point>> andrew(vector<Point> &q) {
    int n = q.size(), top1 = 0, top2 = 0;
    vector<int> stkup(n + 1), stkdown(n + 1);
    vector<bool> used(n + 1, 0);
    sort(q.begin(), q.end(), [&](Point a, Point b) {
        if (cmp(a.x, b.x) == 0) return a.y < b.y;
        else return a.x < b.x;
    });
    for (int i = 0; i < n; i++) {
        while (top1 >= 2 && sign(area(q[stkup[top1 - 1]], q[stkup[top1]], q[i])) >= 0) {
            if (sign(area(q[stkup[top1 - 1]], q[stkup[top1]], q[i])) > 0) used[stkup[top1--]] = 0;
            else top1--;
        }
        stkup[++top1] = i;
        used[i] = 1;
    }
    used[0] = 0, used[stkup[top1]] = 0;
    for (int i = 0; i < n; i++) {
        if (used[i]) continue;
```

```
        while (top2 >= 2 && sign(area(q[stkdown[top2 - 1]], q[stkdown[top2]], q[i])) <= 0) top2--;
        stkdown[++top2] = i;
    }
    vector<Point> res1, res2;
    for (int i = 1; i <= top1; i++) res1.push_back(q[stkup[i]]);
    for (int i = 1; i <= top2; i++) res2.push_back(q[stkdown[i]]);
    return {res1, res2};
}
```

判断点是否在凸包内

O(logn) 需保证给定点按逆时针给出，可先求凸包，凸包上的点已按逆时针

```
const long double eps = 1e-8;
i64 area(Point a, Point b, Point c) {
    return (b - a) ^ (c - a);
}
int sign(i64 x) { // 符号函数
    if (abs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}
bool on_segment(Point p, Point a, Point b) {
    return sign((p - a) ^ (p - b)) == 0 && sign((p - a) * (p - b)) <= 0;
}
// -1 是在边界，1 是在凸包内，0 是在凸包外
int check(i64 x, i64 y, vector<Point> &p) {
    Point u = {x, y};
    if (p.size() == 1) {
        if (u.x == p[0].x && u.y == p[0].y) return -1;
        else return 0;
    }
    if (p.size() == 2) {
        if (on_segment(u, p[0], p[1])) return -1;
        else return 0;
    }
    if (u.x == p[0].x && u.y == p[0].y) return -1;
    if (area(p[0], p[1], u) < 0 || area(p[0], p.back(), u) > 0) return 0;
    auto toleft = [&](Point a, Point b) -> int {
        auto t = a ^ b;
        return (t > eps) - (t < -eps);
    };
    const auto cmp = [&](const Point &u, const Point &v) {return toleft(u - p[0], v - p[0]) == 1;};
    const size_t i = lower_bound(p.begin() + 1, p.end(), u, cmp) - p.begin();
    if (i == 1) {
        if (on_segment(u, p[0], p[i])) return -1;
        else return 0;
    }
    if (i == p.size() - 1 && on_segment(u, p[0], p[i])) return -1;
    if (on_segment(u, p[i - 1], p[i])) return -1;
    return toleft(p[i] - p[i - 1], u - p[i - 1]) > 0;
}
```

求平行于给定直线的凸多边形的切线

返回切点下标 复杂度 O(logn)

```
pair<size_t, size_t> tangent(const vector<Point>& p, const Line& a) {
    const size_t i = extreme(p, [&](const Point& u) { return a.v; });
    const size_t j = extreme(p, [&](const Point& u) { return -a.v; });
    return {i, j};
}
```

判断直线与凸包是否相交

1代表相交 a b 是切点，st ed 是给定直线上两点

```
bool check(Point a, Point b, Point st, Point ed) {
    return sign(((st - a) ^ (ed - a)) * ((st - b) ^ (ed - b))) <= 0;
}
```

过凸多边形外一点求凸多边形的切线

返回切点下标 复杂度 O(logn) 必须保证点在多边形外

```
pair<size_t, size_t> tangent(const vector<Point>& p, const Point& a) {
    const size_t i = extreme(p, [&](const Point& u) { return u - a; });
    const size_t j = extreme(p, [&](const Point& u) { return a - u; });
    return {i, j};
}
```

闵可夫斯基和 判断凸包和凸包的关系

```
struct Segment {
    Point a, b;
};
struct argcmp {
    bool operator()(const Point &a, const Point &b) const {
        const auto quad = [](const Point &a) {
            if (a.y < -eps) return 1;
            if (a.y > eps) return 4;
            if (a.x < -eps) return 5;
            if (a.x > eps) return 3;
            return 2;
        };
        const int qa = quad(a), qb = quad(b);
        if (qa != qb) return qa < qb;
        const auto t = a ^ b;
        if (abs(t) <= eps) return a * a < b * b - eps;  // 不同长度的向量需要分开
        return t > eps;
    }
};
vector<Point> operator +(const vector<Point> &a, const vector<Point> &b) {
    vector<Segment> e1(a.size()), e2(b.size()), edge(a.size() + b.size());
    vector<Point> res; res.reserve(a.size() + b.size());
    const auto cmp = [](const Segment &u, const Segment &v) {return argcmp()(u.b - u.a, v.b - v.a);};
    for (size_t i = 0; i < a.size(); i++) e1[i] = {a[i], a[(i + 1) % a.size()]};
    for (size_t i = 0; i < b.size(); i++) e2[i] = {b[i], b[(i + 1) % b.size()]};
    rotate(e1.begin(), min_element(e1.begin(), e1.end(), cmp), e1.end());
    rotate(e2.begin(), min_element(e2.begin(), e2.end(), cmp), e2.end());
    merge(e1.begin(), e1.end(), e2.begin(), e2.end(), edge.begin(), cmp);
    auto toleft = [&](Point a, Point b) -> int {
        auto t = a ^ b;
        return (t > eps) - (t < -eps);
    };
    const auto check = [&](const vector<Point> &res, const Point &u) {
        const auto back1 = res.back(), back2 = *prev(res.end(), 2);
        return toleft(back1 - back2, u - back1) == 0 && (back1 - back2) * (u - back1) >= -eps;
    };
    auto u = e1[0].a + e2[0].a;
    for (const auto &v : edge) {
        while (res.size() > 1 && check(res,u)) res.pop_back();
        res.push_back(u);
        u = u + v.b - v.a;
    }
    if (res.size() > 1 && check(res, res[0])) res.pop_back();
    return res;
}
```

凸包新加 k 个点后新凸包的面积

```
const int N = 2e5 + 10;
int n, m;
struct Point {
    i64 x, y;
    int id = -1;
    Point(i64 x = 0, i64 y = 0) : x(x), y(y) {}
    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);
    }
    Point operator -() const { return {-x, -y}; }
    i64 operator *(const Point &b) const {
        return x * b.x + y * b.y;
    }
    // 叉积 向量 A 与 B 张成的平行四边形的有向面积
    // P ^ Q > 0, P 在 Q 的顺时针方向; < 0, P 在 Q 的逆时针方向; = 0, P, Q 共线, 可能同向或反向
    i64 operator ^(const Point &b) const {
        return x * b.y - b.x * y;
    }
};
struct Line {
```

```
    Point p, v;
};
const long double eps = 1e-8;
i64 area(Point a, Point b, Point c) {
    return (b - a) ^ (c - a);
}
int sign(i64 x) { // 符号函数
    if (abs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}
bool on_segment(Point p, Point a, Point b) {
    return sign((p - a) ^ (p - b)) == 0 && sign((p - a) * (p - b)) <= 0;
}
int check(i64 x, i64 y, vector<Point> &p) {
    Point u = {x, y};
    if (p.size() == 1) {
        if (u.x == p[0].x && u.y == p[0].y) return -1;
        else return 0;
    }
    if (p.size() == 2) {
        if (on_segment(u, p[0], p[1])) return -1;
        else return 0;
    }
    if (u.x == p[0].x && u.y == p[0].y) return -1;
    if (area(p[0], p[1], u) < 0 || area(p[0], p.back(), u) > 0) return 0;
    auto toleft = [&](Point a, Point b) -> int {
        auto t = a ^ b;
        return (t > eps) - (t < -eps);
    };
    const auto cmp = [&](const Point &u, const Point &v) {return toleft(u - p[0], v - p[0]) == 1;};
    const size_t i = lower_bound(p.begin() + 1, p.end(), u, cmp) - p.begin();
    if (i == 1) {
        if (on_segment(u, p[0], p[i])) return -1;
        else return 0;
    }
    if (i == p.size() - 1 && on_segment(u, p[0], p[i])) return -1;
    if (on_segment(u, p[i - 1], p[i])) return -1;
    return toleft(p[i] - p[i - 1], u - p[i - 1]) > 0;
}
size_t extreme(const vector<Point>& p, const function<Point(const Point&)>& dir) {
    auto toleft = [&](Point a, Point b) -> int {
        auto t = a ^ b;
        return (t > eps) - (t < -eps);
    };
    int len = p.size();
    const auto check = [&](const size_t i) { return toleft(dir(p[i]), p[(i + 1) % len] - p[i]) >= 0; };
    const auto dir0 = dir(p[0]);
    const auto check0 = check(0);
    if (!check0 && check(len - 1)) return 0;
    const auto cmp = [&](const Point& v) {
        const size_t vi = &v - p.data();
        if (vi == 0) return 1;
        const auto checkv = check(vi);
        const auto t = toleft(dir0, v - p[0]);
        if (vi == 1 && checkv == check0 && t == 0) return 1;
        return checkv ^ (checkv == check0 && t <= 0);
    };
    return partition_point(p.begin(), p.end(), cmp) - p.begin();
}
pair<size_t, size_t> tangent(const vector<Point>& p, const Point& a) {
    const size_t i = extreme(p, [&](const Point& u) { return u - a; });
    const size_t j = extreme(p, [&](const Point& u) { return a - u; });
    return {i, j};
}
void solve() {
    cin >> n;
    vector<Point> q(n);
    for (int i = 0; i < n; i++) cin >> q[i].x >> q[i].y;
    auto point = andrew(q);
    i64 sum = 0;
    vector<i64> pre(point.size(), 0);
    for (int i = 0; i < point.size(); i++) {
        int la = i - 1;
        if (la == -1) la = point.size() - 1;
        sum += point[la] ^ point[i];
        pre[i] = sum;
    }
    // 求凸包内靠外的面积
    auto query_sum = [&](int l, int r) -> i64 {
        if (l <= r) return pre[r] - pre[l] + (point[r] ^ point[l]);
```

```
        return pre[point.size() - 1] - pre[l] + pre[r] + (point[r] ^ point[l]);
    };
    auto get_area = [&](vector<Point> &tmp) -> i64 {
        i64 res = 0;
        for (int i = 0; i < tmp.size(); i++) {
            int j = (i + 1) % tmp.size();
            if (tmp[i].id != -1 && tmp[j].id != -1) {
                res += query_sum(tmp[i].id, tmp[j].id);
            }
        }
        return res;
    };
    cin >> m;
    for (int i = 1; i <= m; i++) {
        int k;
        cin >> k;
        vector<Point> tmp;
        for (int j = 0; j < k; j++) {
            i64 x, y;
            cin >> x >> y;
            if (check(x, y, point)) continue;
            auto res = tangent(point, {x, y}); // 求切线
            Point u = point[res.first], v = point[res.second];
            u.id = res.first, v.id = res.second;
            tmp.push_back({x, y}), tmp.push_back(u), tmp.push_back(v);
        }
        if (tmp.size() == 0) {
            cout << sum << '\n';
            continue;
        }
        auto p = andrew(tmp);
        i64 now = 0;
        for (int i = 0; i < p.size(); i++) {
            int ne = (i + 1) % p.size();
            now += p[i] ^ p[ne];
        }
        cout << now + get_area(p) << '\n';
    }
}
```

动态凸包

动态加点，查询直线是否与凸包相交，查询点是否在凸包内, 若只查询点是否在凸包内，可以去掉 segment和 e 相关的部分

```
struct Point {
    i64 x, y;
    int id = -1;
    Point(i64 x = 0, i64 y = 0) : x(x), y(y) {}
    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);
    }
    Point operator -() const { return {-x, -y}; }
    i64 operator *(const Point &b) const {
        return x * b.x + y * b.y;
    }
    Point operator *(const i64 &t) const {
        return {x * t, y * t};
    }
    Point operator /(const i64 &t) const {
        return {x / t, y / t};
    }
    // 叉积 向量 A 与 B 张成的平行四边形的有向面积
    // P ^ Q > 0, P 在 Q 的顺时针方向; < 0, P 在 Q 的逆时针方向; = 0, P, Q 共线, 可能同向或反向
    i64 operator ^(const Point &b) const {
        return x * b.y - b.x * y;
    }
};
const int N = 1e5 + 10;
const long double eps = 1e-8;
int sign(i64 x) { // 符号函数
    if (abs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}
bool argcmp(const Point &a, const Point &b) {
    auto quad = [](const Point &a) {
        if (a.y < -eps) return 1;
```

```
        if (a.y > eps) return 4;
        if (a.x < -eps) return 5;
        if (a.x > eps) return 3;
        return 2;
    };
    const int qa = quad(a), qb = quad(b);
    if (qa != qb) return qa < qb;
    const auto t = a ^ b;
    if (abs(t) <= eps) return a * a < b * b - eps;
    return t > eps;
}
struct Segment {
    Point a, b;
    Segment() = default;
    Segment(Point a, Point b): a(a), b(b) {}
    bool is_on(const Point &c) const {return sign((c - a) ^ (c - b)) == 0 && sign((c - a) * (c - b)) <= 0;}
};
struct Convex {
    set<Point, decltype(&argcmp)> p{&argcmp}; // 坐标扩大三倍，便于整数运算
    map<Point, decltype(p.begin()), decltype(&argcmp)> e{&argcmp};
    Point o = Point(0,0); // 凸包内一点
    inline auto nxt(decltype(p.begin()) it) const {it++; return it == p.end() ? p.begin() : it;}
    inline auto pre(decltype(p.begin()) it) const {if (it == p.begin()) it = p.end(); return --it;}
    bool is_in(const Point &a) const {
        if (p.size() <= 0) return false;
        auto it = p.lower_bound(a * 3 - o);
        if (it == p.end()) it = p.begin();
        return ((*it - *pre(it)) ^ ((a * 3 - o) - *pre(it))) >= -eps;
    }
    bool check(i64 a, i64 b, i64 c) {
        Point lv(b, -a);
        auto cal = [=](Point u) {u = (u + o) / 3; return a * u.x + b * u.y - c;};
        auto sgn = [](i64 x) {return (x > 0) - (x < 0);};
        if (p.size() == 1) {
            Point u = *p.begin();
            if (cal(u) == 0) return false;
            return true;
        }
        if (p.size() == 2) {
            Point u = *p.begin(), v = *p.rbegin();
            if (cal(u) == 0 || cal(v) == 0) return false;
            return sgn(cal(u)) == sgn(cal(v));
        }
        auto eit1 = e.lower_bound(lv), eit2 = e.lower_bound(-lv);
        if (eit1 == e.end()) eit1 = e.begin();
        if (eit2 == e.end()) eit2 = e.begin();
        auto it1 = eit1->second, it2 = eit2->second;
        if (cal(*it1) == 0 || cal(*it2) == 0) return false;
        return sgn(cal(*it1)) == sgn(cal(*it2));
    }
    void insert(Point a) {
        if (p.size() <= 1) {
            p.insert(a * 3);
            return;
        }
        if (p.size() == 2) {
            Point u = *p.begin(), v = *p.rbegin();
            if (Segment(u, v).is_on(a * 3)) return;
            if (Segment(u, a * 3).is_on(v)) {
                p.erase(v); p.insert(a * 3);
                return;
            }
            if (Segment(v, a * 3).is_on(u)) {
                p.erase(u); p.insert(a * 3);
                return;
            }
            o = (u + v + a * 3) / 3;
            p.clear();
            p.insert(u - o); p.insert(v - o); p.insert(a * 3 - o);
            for (auto it = p.begin(); it != p.end(); it++) {e[*nxt(it) - *it] = it;}
            return;
        }
        if (is_in(a)) return;
        a = a * 3 - o;
        auto _it = p.insert(a).first;
        e.erase(*nxt(_it) - *pre(_it));
        auto it = nxt(_it);
        while (p.size() > 3 && ((*it - a) ^ (*nxt(it) - *it)) <= eps) {
            e.erase(*nxt(it) - *it);
            p.erase(it); it = nxt(_it);
        }
```

```
        it = pre(_it);
        while (p.size() > 3 && ((*it - *pre(it)) ^ (a - *it)) <= eps) {
            e.erase(*it - *pre(it));
            p.erase(it); it = pre(_it);
        }
        e[*nxt(_it) - *_it] = _it; e[*_it - *pre(_it)] = pre(_it);
    }
};
void solve() {
    int n, q;
    cin >> n >> q;
    Convex c;
    for (int i = 1; i <= n; i++) {
        i64 x, y;
        cin >> x >> y;
        c.insert({x, y});
    }
    while (q--) {
        int op;
        cin >> op;
        if (op == 1) {
            i64 x, y;
            cin >> x >> y;
            c.insert({x, y});
        }
        else {
            i64 x, y, z;
            cin >> x >> y >> z;
            // ax + by == c
            if (c.check(x, y, z)) cout << "YES\n";
            else cout << "NO\n";
        }
    }
}
```

旋转卡壳

平面最远点对

凸包直径 特判直线

```
const int N = 1e5 + 10;
const long double eps = 1e-8;
int sign(i64 x) { // 符号函数
    if (abs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}
i64 area(Point a, Point b, Point c) {
    return (b - a) ^ (c - a);
}
i64 get_dist(Point a, Point b) {
    i64 dx = a.x - b.x;
    i64 dy = a.y - b.y;
    return dx * dx + dy * dy;
}
i64 rotating_calipers(vector<Point> &p) {
    int n = p.size();
    i64 res = 0;
    for (int i = 0, j = 2; i < n; i++) {
        auto d = p[i], e = p[(i + 1) % n];
        while (area(d, e, p[j]) < area(d, e, p[j + 1])) j = (j + 1) % n;
        res = max(res, max(get_dist(p[j], d), get_dist(p[j], e)));
    }
    return res;
}
void solve() {
    int n;
    cin >> n;
    vector<Point> q(n);
    for (int i = 0; i < n; i++) cin >> q[i].x >> q[i].y;
    auto p = andrew(q);
    if (p.size() == 1) {
        cout << get_dist(q[0], q[n - 1]) << '\n';
        return;
    }
    cout << rotating_calipers(p) << '\n';
}
```

最小矩形覆盖

如果存在面积为 0 特判

```
const int N = 5e4 + 10;
const long double eps = 1e-8, PI = acos(-1);
Point ans[4];
long double min_area = 1e18;
int sign(long double x) { // 符号函数
    if (abs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}
int dcmp(long double x, long double y) {
    if (abs(x - y) < eps) return 0;
    else if (x > y) return 1;
    else return -1;
}
long double area(Point a, Point b, Point c) {
    return (b - a) ^ (c - a);
}
long double get_length(Point a) {
    return sqrtl(a * a);
}
long double get_dist(Point a, Point b) {
    long double dx = a.x - b.x;
    long double dy = a.y - b.y;
    return sqrtl(dx * dx + dy * dy);
}
long double project(Point a, Point b, Point c) { //投影
    return (b - a) * (c - a) / get_length(b - a);
}
Point norm(Point a) { //单位向量
    return a / get_length(a);
}
Point rotate(Point a, long double angle) {
    return Point(a.x * cos(angle) + a.y * sin(angle), -a.x * sin(angle) + a.y * cos(angle));
}
void rotating_calipers(vector<Point> &p) {
    int n = p.size();
    for (int i = 0, a = 2, b = 1, c = 2; i < n; i++) {
        auto d = p[i], e = p[(i + 1) % n];
        while (dcmp(area(d, e, p[a]), area(d, e, p[(a + 1) % n])) <= 0) a = (a + 1) % n;
        while (dcmp(project(d, e, p[b]), project(d, e, p[(b + 1) % n])) <= 0) b = (b + 1) % n;
        if (!i) c = a;
        while (dcmp(project(d, e, p[c]), project(d, e, p[(c + 1) % n])) >= 0) c = (c + 1) % n;
        auto x = p[a], y = p[b], z = p[c];
        auto h = area(d, e, x) / get_length(e - d);
        auto w = (y - z) * (e - d) / get_length(e - d);
        if (h * w < min_area) {
            min_area = h * w;
            ans[0] = d + norm(e - d) * project(d, e, y);
            ans[3] = d + norm(e - d) * project(d, e, z);
            auto u = rotate(norm(e - d), -PI / 2);
            ans[1] = ans[0] + u * h;
            ans[2] = ans[3] + u * h;
        }
    }
}
void solve() {
    int n;
    cin >> n;
    vector<Point> q(n);
    for (int i = 0; i < n; i++) cin >> q[i].x >> q[i].y;
    auto p = andrew(q);
    rotating_calipers(p);
    int k = 0;
    for (int i = 1; i < 4; i++) {
        if (dcmp(ans[i].y, ans[k].y) < 0 || !dcmp(ans[i].y, ans[k].y) && dcmp(ans[i].x, ans[k].x) < 0) {
            k = i;
        }
    }
    cout << fixed << setprecision(5) << min_area << '\n';
    for (int i = 0; i < 4; i++, k++) {
        auto t1 = ans[k % 4].x, t2 = ans[k % 4].y;
        if (!sign(t1)) t1 = 0;
        if (!sign(t2)) t2 = 0;
        cout << fixed << setprecision(5) << t1 << ' ' << t2 << '\n';
    }
}
```

半平面交

排序增量法，复杂度 O(nlogn)

输入与返回值都是用直线表示的半平面集合

```
vector<Line> halfinter(vector<Line> l, const long double lim = 1e9) {
    const auto check = [](const Line &a, const Line &b, const Line &c) {
        return a.toleft(b.inter(c)) < 0;
    };
    // 无精度误差的方法，但注意取值范围会扩大到三次方
    /*const auto check = [](const Line &a, const Line &b, const Line &c) {
        const Point p = a.v * (b.v ^ c.v), q = b.p * (b.v ^ c.v) + b.v * (c.v ^ (b.p - c.p)) - a.p * (b.v ^ c.v);
        return p.toleft(q) < 0;
    };*/
    l.push_back({{-lim, 0}, {0, -1}}); l.push_back({{0, -lim}, {1, 0}});
    l.push_back({{lim, 0}, {0, 1}}); l.push_back({{0, lim}, {-1, 0}});
    sort(l.begin(), l.end());
    deque<Line> q;
    for (size_t i = 0; i < l.size(); i++) {
        if (i > 0 && l[i - 1].v.toleft(l[i].v) == 0 && l[i - 1].v * l[i].v > eps) continue;
        while (q.size() > 1 && check(l[i], q.back(), q[q.size() - 2])) q.pop_back();
        while (q.size() > 1 && check(l[i], q[0], q[1])) q.pop_front();
        if (!q.empty() && q.back().v.toleft(l[i].v) <= 0) return vector<Line>();
        q.push_back(l[i]);
    }
    while (q.size() > 1 && check(q[0], q.back(), q[q.size() - 2])) q.pop_back();
    while (q.size() > 1 && check(q.back(), q[0], q[1])) q.pop_front();
    return vector<Line>(q.begin(), q.end());
}
```

凸多边形内的最大内切圆半径

二分半径，然后将凸多边形向内缩即可

```
void solve() {
    int n;
    cin >> n;
    vector<Point> q(n);
    for (int i = 0; i < n; i++) cin >> q[i].x >> q[i].y;
    auto check = [&](long double r) -> bool {
        vector<Line> lines;
        for (size_t i = 0; i < n; i++) {
            Line l = {q[i], q[(i + 1) % n] - q[i]};
            const Point v{-l.v.y, l.v.x}; // 垂直方向
            const Point t = l.p + v / v.len() * r; // 平移后的点
            lines.push_back({t, l.v});
        }
        const auto halfs = halfinter(lines);
        return !halfs.empty();
    };
    long double l = 0, r = 20000;
    while (r - l > eps) {
        long double mid = (l + r) / 2;
        if (check(mid)) l = mid;
        else r = mid;
    }
    cout << fixed << setprecision(10) << l << '\n';
}
```

面积并

多边形面积并

需保证每个多边形的点是逆时针的

轮廓积分，复杂度 O(n^2logn)，n 为边数

ans[i] 表示被至少覆盖了 i + 1 次的区域的面积

```
vector<long double> area_union(const vector<vector<Point>> &polys) {
    const size_t siz = polys.size();
    vector<vector<pair<Point, Point>>> segs(siz);
    const auto check = [](const Point &u, const Segment &e) {
        return !((u < e.a && u < e.b) || (u > e.a && u > e.b));
    };
    auto cut_edge = [&](const Segment &e, const size_t i) {
        const Line le{e.a, e.b - e.a};
        vector<pair<Point, int>> evt;
        evt.push_back({e.a, 0}); evt.push_back({e.b, 0});
        for (size_t j = 0; j < polys.size(); j++) {
```

```
            if (i == j) continue;
            const auto &pj = polys[j];
            for (size_t k = 0; k < pj.size(); k++) {
                const Segment s = {pj[k], pj[(k + 1) % pj.size()]};
                if (le.toleft(s.a) == 0 && le.toleft(s.b) == 0) {
                    evt.push_back({s.a, 0});
                    evt.push_back({s.b, 0});
                }
                else if (s.is_inter(le)) {
                    const Line ls{s.a, s.b - s.a};
                    const Point u = le.inter(ls);
                    if (le.toleft(s.a) < 0 && le.toleft(s.b) >= 0) evt.push_back({u, -1});
                    else if (le.toleft(s.a) >= 0 && le.toleft(s.b) < 0) evt.push_back({u, 1});
                }
            }
        }
        sort(evt.begin(), evt.end());
        if (e.a > e.b) reverse(evt.begin(), evt.end());
        int sum = 0;
        for (size_t i = 0; i < evt.size(); i++) {
            sum += evt[i].second;
            const Point u = evt[i].first, v = evt[i + 1].first;
            if (!(u == v) && check(u, e) && check(v, e)) segs[sum].push_back({u, v});
            if (v == e.b) break;
        }
    };
    for (size_t i = 0; i < polys.size(); i++) {
        const auto &pi = polys[i];
        for (size_t k = 0; k < pi.size(); k++) {
            const Segment ei = {pi[k], pi[(k + 1) % pi.size()]};
            cut_edge(ei, i);
        }
    }
    vector<long double> ans(siz);
    for (size_t i = 0; i < siz; i++) {
        long double sum = 0;
        sort(segs[i].begin(), segs[i].end());
        int cnt = 0;
        for (size_t j = 0; j < segs[i].size(); j++) {
            if (j > 0 && segs[i][j] == segs[i][j - 1]) segs[i + (++cnt)].push_back(segs[i][j]);
            else cnt = 0, sum += segs[i][j].first ^ segs[i][j].second;
        }
        ans[i] = sum / 2;
    }
    return ans;
}
```

矩形面积并 O(nlogn)

输入为 n 个矩形的左上角坐标和右下角坐标, 坐标为矩阵的系，非笛卡尔坐标系

如果是笛卡尔坐标系可以直接交换输入的 x 和 y

```
int n;
struct Segment {
    long double x, y, yy;
    int k;
    bool operator< (const Segment &t) const {
        return x < t.x;
    }
} seg[N * 2];
struct Node {
    int l, r, cnt;
    long double len;
} tr[N * 8];
vector<long double> ys;
int find(long double x) {
    return lower_bound(ys.begin(), ys.end(), x) - ys.begin();
}
void puhsup(int u) {
    if (tr[u].cnt) tr[u].len = ys[tr[u].r + 1] - ys[tr[u].l];
    else if (tr[u].l != tr[u].r) tr[u].len = tr[u << 1].len + tr[u << 1 | 1].len;
    else if (tr[u].l == tr[u].r) tr[u].len = 0;
}
void build(int u, int l, int r) {
    tr[u] = {l, r, 0, 0};
    if (l != r) {
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
    }
```

```
}
void modify(int u, int l, int r, int k) {
    if (tr[u].l >= l && tr[u].r <= r) {
        tr[u].cnt += k;
        puhsup(u);
    }
    else {
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid) modify(u << 1, l, r, k);
        if (r > mid) modify(u << 1 | 1, l, r, k);
        puhsup(u);
    }
}
void solve() {
    cin >> n;
    for (int i = 0, j = 0; i < n; i++) {
        long double x, y, xx, yy;
        cin >> x >> y >> xx >> yy;
        seg[j++] = {x, y, yy, 1};
        seg[j++] = {xx, y, yy, -1};
        ys.push_back(y), ys.push_back(yy);
    }
    sort(ys.begin(), ys.end());
    ys.erase(unique(ys.begin(), ys.end()), ys.end());
    build(1, 0, ys.size() - 2);
    sort(seg, seg + n * 2);
    long double res = 0;
    for (int i = 0; i < n * 2; i++) {
        if (i > 0) res += tr[1].len * (seg[i].x - seg[i - 1].x);
        modify(1, find(seg[i].y), find(seg[i].yy) - 1, seg[i].k);
    }
    cout << fixed << setprecision(2);
    cout << res << '\n';
}
```

面积最大、最小三角形

面积最大三角形

面积最大可以通过旋转卡壳做到 O(n^2)

```
i64 get_mx(vector<Point> &p) { // p是凸包
    if (p.size() <= 2) return 0;
    int n = p.size();
    i64 ans = 0;
  for (int i = 0; i < n; i++) {
      int j = (i + 1) % n;
      int k = (j + 1) % n;
      while (j != i && k != i) {
            ans = max(ans, abs((p[j] - p[i]) ^ (p[k] - p[i])));
            while (((p[i] - p[j]) ^ (p[(k + 1) % n] - p[k])) < 0) k = (k + 1) % n;
            j = (j + 1) % n;
        }
  }
  return ans;
}
```

极角序扫描线，复杂度 O(n^2logn)

记得重载等于号和小于号

```
struct argcmp {
    bool operator()(const Point &a, const Point &b) const {
        const auto quad = [](const Point &a) {
            if (a.y < -eps) return 1;
            if (a.y > eps) return 4;
            if (a.x < -eps) return 5;
            if (a.x > eps) return 3;
            return 2;
        };
        const int qa = quad(a), qb = quad(b);
        if (qa != qb) return qa < qb;
        const auto t = a ^ b;
        if (abs(t) <= eps) return a * a < b * b - eps;  // 不同长度的向量需要分开
        return t > eps;
    }
};
pair<i64, i64> minmax_triangle(const vector<Point> &vec) {
    if (vec.size() <= 2) return {0, 0};
    vector<pair<int, int>> evt;
```

```
    evt.reserve(vec.size() * vec.size());
    i64 maxans = 0, minans = numeric_limits<i64>::max();
    for (size_t i = 0; i < vec.size(); i++) {
        for (size_t j = 0; j < vec.size(); j++) {
            if (i == j) continue;
            if (vec[i] == vec[j]) minans = 0;
            else evt.push_back({i, j});
        }
    }
    sort(evt.begin(), evt.end(), [&](const pair<int, int> &u, const pair<int, int> &v) {
        const Point du = vec[u.second] - vec[u.first], dv = vec[v.second] - vec[v.first];
        return argcmp()({du.y, -du.x}, {dv.y, -dv.x});
    });
    vector<size_t> vx(vec.size()), pos(vec.size());
    for (size_t i = 0; i < vec.size(); i++) vx[i] = i;
    sort(vx.begin(), vx.end(), [&](int x, int y) {return vec[x] < vec[y];});
    for (size_t i = 0; i < vx.size(); i++) pos[vx[i]] = i;
    for (auto [u, v] : evt) {
        const size_t i = pos[u], j = pos[v];
        const size_t l = min(i, j), r = max(i, j);
        const Point vecu = vec[u], vecv = vec[v];
        if (l > 0) minans = min(minans, abs((vec[vx[l - 1]] - vecu) ^ (vec[vx[l - 1]] - vecv)));
        if (r < vx.size() - 1) minans = min(minans, abs((vec[vx[r + 1]] - vecu) ^ (vec[vx[r + 1]] - vecv)));
        maxans = max({maxans, abs((vec[vx[0]] - vecu) ^ (vec[vx[0]] - vecv)), abs((vec[vx.back()] - vecu) ^
(vec[vx.back()] - vecv))});
        if (i < j) swap(vx[i], vx[j]), pos[u] = j, pos[v] = i;
    }
    return {minans, maxans};
}
```

判断多个线段是否存在交点

扫描线，复杂度 O(nlogn)

```
bool segs_inter(const vector<Segment> &segs) {
    if (segs.empty()) return false;
    using seq_t = tuple<long double, int, Segment>;
    const auto seqcmp = [](const seq_t &u, const seq_t &v) {
        const auto [u0, u1, u2] = u;
        const auto [v0, v1, v2] = v;
        if (abs(u0 - v0) <= eps) return make_pair(u1, u2) < make_pair(v1, v2);
        return u0 < v0 - eps;
    };
    vector<seq_t> seq;
    for (auto seg : segs) {
        if (seg.a.x > seg.b.x + eps) swap(seg.a, seg.b);
        seq.push_back({seg.a.x, 0, seg});
        seq.push_back({seg.b.x, 1, seg});
    }
    sort(seq.begin(), seq.end(), seqcmp);
    long double x_now;
    auto cmp = [&](const Segment &u, const Segment &v) {
        if (abs(u.a.x - u.b.x) <= eps || abs(v.a.x - v.b.x) <= eps) return u.a.y < v.a.y - eps;
        return ((x_now - u.a.x) * (u.b.y - u.a.y) + u.a.y * (u.b.x - u.a.x)) * (v.b.x - v.a.x) < ((x_now - v.a.x) *
(v.b.y - v.a.y) + v.a.y * (v.b.x - v.a.x)) * (u.b.x - u.a.x) - eps;
    };
    multiset<Segment, decltype(cmp)> s{cmp};
    for (const auto [x, o, seg] : seq) {
        x_now = x;
        const auto it = s.lower_bound(seg);
        if (o == 0) {
            if (it != s.end() && seg.is_inter(*it)) return true;
            if (it != s.begin() && seg.is_inter(*prev(it))) return true;
            s.insert(seg);
        }
        else {
            if (next(it) != s.end() && it != s.begin() && (*prev(it)).is_inter(*next(it))) return true;
            s.erase(it);
        }
    }
    return false;
}
```

平面最近点对

```
i64 power2(i64 a) {
    return a * a;
}
i64 dist(pair<i64, i64> a, pair<i64, i64> b) {
```

```
    i64 dx = a.first - b.first;
    i64 dy = a.second - b.second;
    return power2(dx) + power2(dy);
}
pair<i64, i64> rev(pair<i64, i64> a) {
    return {a.second, a.first};
}
void solve() {
    int n;
    cin >> n;
    vector<pair<i64, i64>> p(n + 1);
    set<pair<i64, i64>> S;
    i64 D;
    for (int i = 0; i < n; i++) cin >> p[i].first >> p[i].second;
    D = dist(p[0], p[1]);
    sort(p.begin(), p.begin() + n);
    for (int tt = 0, hh = 0; tt < n; tt++) {
        pair<i64, i64> ver = rev(p[tt]);
        while (D && D <= power2((i64)(p[tt].first - p[hh].first))) S.erase(rev(p[hh++]));
        auto it = S.lower_bound(ver);
        for (auto i = it; i != S.end() && power2(i->first - ver.first) < D; i++) D = min(D, dist(ver, *i));
        if (it != S.begin()) {
            for (auto i = --it; power2(i->first - ver.first) < D; i--) {
                D = min(D, dist(ver, *i));
                if (i == S.begin()) break;
            }
        }
        S.insert(ver);
    }
    cout << D << '\n';
}
```

三维凸包

```
const int N = 2200, M = 220, mod = 998244353, P = 110;
int n, m; // 点数，面数
bool g[N][N]; // 表示两点逆时针构成的面是否被照射到
const long double eps = 1e-13;
long double rand_eps() {
    return ((long double)rand() / RAND_MAX - 0.5) * eps;
}
struct Point {
    long double x, y, z;
    void shake() { // 给每个点随机扰动，防止出现四点共面
        x += rand_eps(), y += rand_eps(), z += rand_eps();
    }
    Point operator- (Point t) {
        return {x - t.x, y - t.y, z - t.z};
    }
    long double operator& (Point t) { // 点积
        return x * t.x + y * t.y + z * t.z;
    }
    Point operator* (Point t) {
        return {y * t.z - t.y * z, z * t.x - x * t.z, x * t.y - y * t.x};
    }
    long double len() {
        return sqrtl(x * x + y * y + z * z);
    }
} q[N];
struct Plane {
    int v[3];
    Point norm() { // 法向量
        return (q[v[1]] - q[v[0]]) * (q[v[2]] - q[v[0]]);
    }
    long double area() {
        return norm().len() / 2;
    }
    bool above(Point a) {
        return ((a - q[v[0]]) & norm()) >= 0;
    }
} plane[N * 2], np[N * 2]; // np用来备份
void get_convex_3d() {
    plane[m++] = {0, 1, 2};
    plane[m++] = {2, 1, 0};
    for (int i = 3; i < n; i++) {
        int cnt = 0;
        for (int j = 0; j < m; j++) {
            bool t = plane[j].above(q[i]);
            if (!t) np[cnt++] = plane[j];
            for (int k = 0; k < 3; k++) g[plane[j].v[k]][plane[j].v[(k + 1) % 3]] = t;
```

```
        }
        for (int j = 0; j < m; j++) {
            for (int k = 0; k < 3; k++) {
                int a = plane[j].v[k], b = plane[j].v[(k + 1) % 3];
                if (g[a][b] && !g[b][a]) np[cnt++] = {a, b, i};
            }
        }
        m = cnt;
        for (int j = 0; j < m; j++) plane[j] = np[j];
    }
}
void solve() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> q[i].x >> q[i].y >> q[i].z;
        q[i].shake();
    }
    get_convex_3d();
    double res = 0;
    for (int i = 0; i < m; i++) res += plane[i].area();
    cout << fixed << setprecision(6) << res;
}
```

线段与多边形公共部分的长度

```
bool inside(Point a, Point b, Point c) {
    long double A = (a - b) * (a - c);
    long double B = (a - b) ^ (a - c);
    if (abs(B) > eps) return 0;
    return A <= 0;
}
int intersect(Point a, Point b, Point c, Point d, Point &res) {
    if (abs((b - a) ^ (d - c)) < eps) {
    if (abs((b - c) ^ (a - c)) < eps) return -1;
    return 0;
    }
    long double chk1 = (b - a) ^ (c - a);
    long double chk2 = (b - a) ^ (d - a);
    if (chk1 > eps && chk2 > eps) return 0;
    if (chk1 < eps && chk2 < eps) return 0;
    long double s1 = (b - a) ^ (d - c), s2 = (b - c) ^ (a - c);
  long double s = s2 / s1;
  res = c + ((d - c) * s);
    return 1;
}
long double sto[N];
pair<long double, long double> V[N];
void solve() {
    int n, m;
    cin >> n >> m;
    vector<Point> p(n);
    for (int i = 0; i < n; i++) cin >> p[i].x >> p[i].y;
    while (m--) {
        long double x, y, xx, yy;
        cin >> x >> y >> xx >> yy;
        Point a = {x, y}, b = {xx, yy};
        int cnt = 0, cv = 0;
    long double len = (b - a) * (b - a);
    for (int i = 0; i < n; i++) {
      Point tp;
      int chk = intersect(a, b, p[i], p[(i + 1) % n], tp);
      if (chk > 0) sto[cnt++] = ((tp - a) * (b - a)) / len;
            else if (chk < 0) {
        long double x = ((p[i] - a) * (b - a)) / len;
        long double y = ((p[(i + 1) % n] - a) * (b - a)) / len;
        if (x > y) swap(x, y);
        V[cv++] = {x, y};
      }
    }
    sort(sto, sto + cnt);
    for (int i = 0; i + 1 < cnt; i += 2) V[cv++] = {sto[i], sto[i + 1]};
    sort(V, V + cv);
    long double rlt = 0, cur = -(1e18);
    len = sqrtl(len);
    for (int i = 0; i < cv; i++) {
      if (cur < V[i].first) cur = V[i].first;
      if (cur < V[i].second) rlt += V[i].second - cur, cur = V[i].second;
    }
        cout << fixed << setprecision(15) << rlt * len << '\n';
    }
```

```
}
```

圆

三角形外接圆

```
Circle CircumscribedCircle(int x1, int y1, int x2, int y2, int x3, int y3) {
    const Point a = {1.0 * x1, 1.0 * y1}, b = {1.0 * x2, 1.0 * y2}, c = {1.0 * x3, 1.0 * y3};
    const Line l1 = {(a + b) / 2, {-(b - a).y, (b - a).x}}, l2 = {(b + c) / 2, {-(c - b).y, (c - b).x}};
    const Point o = l1.inter(l2);
    return Circle{o, o.dis(a)};
}
```

三角形内接圆

```
Circle InscribedCircle(int x1, int y1, int x2, int y2, int x3, int y3) {
    const Point a = {1.0 * x1, 1.0 * y1}, b = {1.0 * x2, 1.0 * y2}, c = {1.0 * x3, 1.0 * y3};
    const Point ab = (b - a) / (b - a).len(), ac = (c - a) / (c - a).len();
    const Line l1 = {a, (ab + ac) / 2};
    const Point ba = (a - b) / (a - b).len(), bc = (c - b) / (c - b).len();
    const Line l2 = {b, (ba + bc) / 2};
    const Point o = l1.inter(l2);
    return Circle{o, Line{a, b - a}.dis(o)};
}
```

通过一点作圆的切线 返回角度

```
vector<long double> TangentLineThroughPoint(int xc, int yc, int r, int xp, int yp) {
    const Circle c = {{1.0 * xc, 1.0 * yc}, 1.0 * r};
    const Point p = {1.0 * xp, 1.0 * yp};
    const auto tans = c.tangent(p);
    vector<long double> ans;
    for (const Line &line : tans) {
        long double th = atan2(line.v.y, line.v.x);
        if (th < -eps) th = PI - abs(th);
        ans.push_back(th / PI * 180);
    }
    sort(ans.begin(), ans.end());
    return ans;
}
```

通过一点并切于一条直线的圆

x1, y1, x2, y2为直线两点 返回圆心

```
vector<Point> CircleThroughAPointAndTangentToALineWithRadius(int xp, int yp, int x1, int y1, int x2, int y2, int r) {
    const Point p = {1.0 * xp, 1.0 * yp}, a = {1.0 * x1, 1.0 * y1}, b = {1.0 * x2, 1.0 * y2};
    const Circle c = {p, 1.0 * r};
    Point d = {-(b - a).y, (b - a).x}; d = d / d.len() * r;
    const Line l1 = {a + d, b - a}, l2 = {a - d, b - a};
    const auto t1 = c.inter(l1), t2 = c.inter(l2);
    auto ans = t1;
    ans.insert(ans.end(), t2.begin(), t2.end());
    sort(ans.begin(), ans.end());
    return ans;
}
```

与两条直线相切的圆 返回圆心

```
vector<Point> CircleTangentToTwoLinesWithRadius(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4, int r)
{
    const Point a = {1.0 * x1, 1.0 * y1}, b = {1.0 * x2, 1.0 * y2}, c = {1.0 * x3, 1.0 * y3}, d = {1.0 * x4, 1.0 * y4};
    Point d1 = {-(b - a).y, (b - a).x}; d1 = d1 / d1.len() * r;
    Point d2 = {-(d - c).y, (d - c).x}; d2 = d2 / d2.len() * r;
    const Line l11 = {a + d1, b - a}, l12 = {a - d1, b - a}, l21 = {c + d2, d - c}, l22 = {c - d2, d - c};
    vector<Point> ans = {l11.inter(l21), l11.inter(l22), l12.inter(l21), l12.inter(l22)};
    sort(ans.begin(), ans.end());
    return ans;
}
```

与两个不相交的圆外切的圆 返回圆心

```
vector<Point> CircleTangentToTwoDisjointCirclesWithRadius(int x1, int y1, int r1, int x2, int y2, int r2, int r) {
    const Point a = {1.0 * x1, 1.0 * y1}, b = {1.0 * x2, 1.0 * y2};
    const Circle c = {a, 1.0 * (r1 + r)}, d = {b, 1.0 * (r2 + r)};
    auto ans = c.inter(d);
    sort(ans.begin(), ans.end());
    return ans;
}
```

圆与多边形面积交 记得输出绝对值

```
long double area_inter(const Circle &circ, const vector<Point> &poly) {
    const auto cal = [](const Circle &circ, const Point &a, const Point &b) {
        if ((a - circ.c).toleft(b - circ.c) == 0) return 0.0l;
        const auto ina = circ.is_in(a), inb = circ.is_in(b);
        const Line ab = {a, b - a};
        if (ina && inb) return ((a - circ.c) ^ (b - circ.c)) / 2;
        if (ina && !inb) {
            const auto t = circ.inter(ab);
            const Point p = t.size() == 1 ? t[0] : t[1];
            const long double ans = ((a - circ.c) ^ (p - circ.c)) / 2;
            const long double th = (p - circ.c).ang(b - circ.c);
            const long double d = circ.r * circ.r * th / 2;
            if ((a - circ.c).toleft(b - circ.c) == 1) return ans + d;
            return ans - d;
        }
        if (!ina && inb) {
            const Point p = circ.inter(ab)[0];
            const long double ans = ((p - circ.c) ^ (b - circ.c)) / 2;
            const long double th = (a - circ.c).ang(p - circ.c);
            const long double d = circ.r * circ.r * th / 2;
            if ((a - circ.c).toleft(b - circ.c) == 1) return ans + d;
            return ans - d;
        }
        const auto p = circ.inter(ab);
        if (p.size() == 2 && Segment{a, b}.dis(circ.c) <= circ.r + eps) {
            const long double ans = ((p[0] - circ.c) ^ (p[1] - circ.c)) / 2;
            const long double th1 = (a - circ.c).ang(p[0] - circ.c), th2 = (b - circ.c).ang(p[1] - circ.c);
            const long double d1 = circ.r * circ.r * th1 / 2, d2 = circ.r * circ.r * th2 / 2;
            if ((a - circ.c).toleft(b - circ.c) == 1) return ans + d1 + d2;
            return ans - d1 - d2;
        }
        const long double th = (a - circ.c).ang(b - circ.c);
        if ((a - circ.c).toleft(b - circ.c) == 1) return circ.r * circ.r * th / 2;
        return -circ.r * circ.r * th / 2;
    };
    long double ans = 0;
    for (size_t i = 0; i < poly.size(); i++) {
        const Point a = poly[i], b = poly[(i + 1) % poly.size()];
        ans += cal(circ, a, b);
    }
    return ans;
}
```

圆面积并 记得输出绝对值

轮廓积分，复杂度 O(n^2logn)

ans[i] 表示被至少覆盖了 i + 1 次的区域的面积

```
vector<long double> area_union(const vector<Circle> &circs) {
    const size_t siz = circs.size();
    using arc_t = tuple<Point, long double, long double, long double>;
    vector<vector<arc_t>> arcs(siz);
    const auto eq = [](const arc_t &u, const arc_t &v) {
        const auto [u1, u2, u3, u4] = u;
        const auto [v1, v2, v3, v4] = v;
        return u1 == v1 && abs(u2 - v2) <= eps && abs(u3 - v3) <= eps && abs(u4 - v4) <= eps;
    };
    auto cut_circ = [&](const Circle &ci, const size_t i) {
        vector<pair<long double, int>> evt;
        evt.push_back({-PI, 0}); evt.push_back({PI, 0});
        int init = 0;
        for (size_t j = 0; j < circs.size(); j++) {
            if (i == j) continue;
            const Circle &cj = circs[j];
            if (ci.r < cj.r - eps && ci.relation(cj) >= 3) init++;
            const auto inters = ci.inter(cj);
            if (inters.size() == 1) evt.push_back({atan2l((inters[0] - ci.c).y, (inters[0] - ci.c).x), 0});
            if (inters.size() == 2) {
```

```
                const Point dl = inters[0] - ci.c, dr = inters[1] - ci.c;
                long double argl = atan2l(dl.y, dl.x), argr = atan2l(dr.y, dr.x);
                if (abs(argl + PI) <= eps) argl = PI;
                if (abs(argr + PI) <= eps) argr = PI;
                if (argl > argr + eps) {
                    evt.push_back({argl, 1}); evt.push_back({PI, -1});
                    evt.push_back({-PI, 1}); evt.push_back({argr, -1});
                }
                else {
                    evt.push_back({argl, 1});
                    evt.push_back({argr, -1});
                }
            }
        }
        sort(evt.begin(), evt.end());
        int sum = init;
        for (size_t i = 0; i < evt.size(); i++) {
            sum += evt[i].second;
            if (abs(evt[i].first - evt[i + 1].first) > eps) arcs[sum].push_back({ci.c, ci.r, evt[i].first, evt[i +
1].first});
            if (abs(evt[i + 1].first - PI) <= eps) break;
        }
    };
    const auto oint = [](const arc_t &arc) {
        const auto [cc, cr, l, r] = arc;
        if (abs(r - l - PI - PI) <= eps) return 2.0l * PI * cr * cr;
        return cr * cr * (r - l) + cc.x * cr * (sinl(r) - sinl(l)) - cc.y * cr * (cosl(r) - cosl(l));
    };
    for (size_t i = 0; i < circs.size(); i++) {
        const auto &ci = circs[i];
        cut_circ(ci, i);
    }
    vector<long double> ans(siz);
    for (size_t i = 0; i < siz; i++) {
        long double sum = 0;
        sort(arcs[i].begin(), arcs[i].end());
        int cnt = 0;
        for (size_t j = 0; j < arcs[i].size(); j++) {
            if (j > 0 && eq(arcs[i][j], arcs[i][j - 1])) arcs[i + (++cnt)].push_back(arcs[i][j]);
            else cnt = 0, sum += oint(arcs[i][j]);
        }
        ans[i] = sum / 2;
    }
    return ans;
}
```

最小圆覆盖

```
Circle get_circle(Point a, Point b, Point c) {
    const Line l1 = {(a + b) / 2, {-(b - a).y, (b - a).x}}, l2 = {(b + c) / 2, {-(c - b).y, (c - b).x}};
    const Point o = l1.inter(l2);
    return Circle{o, o.dis(a)};
}
void solve() {
    int n;
    cin >> n;
    vector<Point> p(n);
    for (int i = 0; i < n; i++) cin >> p[i].x >> p[i].y;
    random_shuffle(p.begin(), p.end());
    Circle c;
    c.c = p[0], c.r = 0;
    for (int i = 1; i < n; i++) {
         if (cmp(c.r, c.c.dis(p[i])) < 0) {
            c.c = p[i], c.r = 0;
            for (int j = 0; j < i; j++) {
                if (cmp(c.r, c.c.dis(p[j])) < 0) {
                    c.c = (p[i] + p[j]) / 2, c.r = p[i].dis(p[j]) / 2;
                    for (int k = 0; k < j; k++) {
                        if (cmp(c.r, c.c.dis(p[k])) < 0) {
                            c = get_circle(p[i], p[j], p[k]);
                        }
                    }
                }
            }
         }
    }
    cout << fixed << setprecision(12) << c.r << '\n';
    cout << fixed << setprecision(12) << c.c.x << ' ' << c.c.y << '\n';
}
```

自适应辛普森积分

```
const long double eps = 1e-12;
long double f(long double x) {

}
long double simpson(long double l, long double r) {
    long double mid = (l + r) / 2;
    return (r - l) * (f(l) + 4 * f(mid) + f(r)) / 6;
}
long double asr(long double l, long double r, long double s) {
    long double mid = (l + r) / 2;
    long double left = simpson(l, mid), right = simpson(mid, r);
    if (fabs(left + right - s) < eps) return left + right;
    else return asr(l, mid, left) + asr(mid, r, right);
}
// asr(l, r, simpson(l, r))
```

树、图

2-SAT

```
int n, m;
int h[N], e[M], ne[M], idx;
int dfn[N], low[N], timestamp;
int stk[N], top;
bool in_stk[N];
int id[N], scc_cnt, sz[N];
void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}
void tarjan(int u) {
    dfn[u] = low[u] = ++timestamp;
    stk[++top] = u, in_stk[u] = 1;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (!dfn[j]) {
            tarjan(j);
            low[u] = min(low[u], low[j]);
        }
        else if (in_stk[j]) low[u] = min(low[u], dfn[j]);
    }
    if (dfn[u] == low[u]) {
        ++scc_cnt;
        int y;
        do {
            y = stk[top--];
            in_stk[y] = 0;
            id[y] = scc_cnt;
            sz[scc_cnt]++;
        } while (y != u);
    }
}
void solve() {
    cin >> n >> m;
    memset(h, -1, sizeof(h));
    while (m--) { // 2i + 0假命题，2i + 1真命题
        int i, a, j, b;
        cin >> i >> a >> j >> b;
        i--, j--;
        add(2 * i + !a, 2 * j + b);
        add(2 * j + !b, 2 * i + a);
    }
    for (int i = 0; i < 2 * n; i++) {
        if (!dfn[i]) {
            tarjan(i);
        }
    }
    for (int i = 0; i < n; i++) {
        if (id[2 * i] == id[2 * i + 1]) {
            cout << "IMPOSSIBLE" << '\n';
            return;
        }
    }
    cout << "POSSIBLE" << '\n';
    for (int i = 0; i < n; i++) {
        if (id[2 * i] < id[2 * i + 1]) cout << 0 << ' ';
        else cout << 1 << ' ';
    }
```

```
}
```

差分约束

求每个变量的最小值用最长路模型，求最大值用最短路模型

最长路模型出现正环无解，最短路模型出现负环无解

如果不存在一个点可以遍历所有边，就需要建一个超级源点

边权有正有负只能差分约束最坏 O(nm), 边权非负则可 tarjan 缩点后拓扑排序 O(n + m), 边权全大于 0 则可直接拓扑排序

最长路

建边 Xi >= Xj + c 从 j 到 i 连一条权值为 c 的边

```
int n, m;
int dist[N], cnt[N], vis[N];
vector<array<int, 2>> adj[N];
bool spfa() {
    memset(dist, -0x3f, sizeof(dist));
    memset(cnt, 0, sizeof(cnt)); // 看情况
    memset(vis, 0, sizeof(vis)); // 看情况
    dist[0] = 0;
    stack<int> q; // 判断无解时用栈，一定有解的时候用队列
    q.push(0);
    vis[0] = 1;
    while (q.size()) {
        auto u = q.top();
        q.pop();
        vis[u] = 0;
        for (auto [v, w] : adj[u]) {
            if (dist[v] < dist[u] + w) {
                cnt[v] = cnt[u] + 1;
                if (cnt[v] >= n + 1) return 1;
                dist[v] = dist[u] + w;
                if (!vis[v]) {
                    vis[v] = 1;
                    q.push(v);
                }
            }
        }
    }
    return 0;
}
void solve() {
    cin >> n >> m;
    for (int i = 0; i <= n; i++) adj[i].clear();
    for (int i = 1; i <= m; i++) {
        int op;
        cin >> op;
        if (op == 1) {
            int a, b, c;
            cin >> a >> b >> c;
            adj[b].push_back({a, c});
        }
        else if (op == 2) {
            int a, b, c;
            cin >> a >> b >> c;
            adj[a].push_back({b, -c});
        }
        else {
            int a, b;
            cin >> a >> b;
            adj[a].push_back({b, 0});
            adj[b].push_back({a, 0});
        }
    }
    for (int i = 1; i <= n; i++) adj[0].push_back({i, 0});
    if (spfa()) cout << -1 << '\n';
    else {
        int ans = 0;
        for (int i = 1; i <= n; i++) ans += dist[i];
        cout << ans << '\n';
    }
}
```

最短路

建边 Xi <= Xj + c 从 j 到 i 连一条权值为 c 的边

```
bool spfa() {
    memset(dist, 0x3f, sizeof(dist));
    memset(cnt, 0, sizeof(cnt)); // 看情况
    memset(vis, 0, sizeof(vis)); // 看情况
    dist[0] = 0;
    stack<int> q; // 判断无解时用栈，一定有解的时候用队列
    q.push(0);
    vis[0] = 1;
    while (q.size()) {
        auto k = q.top();
        q.pop();
        vis[k] = 0;
        for (int i = h[k]; i != -1; i = ne[i]) {
            int j = e[i];
            if (dist[j] > dist[k] + w[i]) {
                cnt[j] = cnt[k] + 1;
                if (cnt[j] >= n + 1) return 1;
                dist[j] = dist[k] + w[i];
                if (!vis[j]) {
                    vis[j] = 1;
                    q.push(j);
                }
            }
        }
    }
    return 0;
}
```

二分图

二分图中： 最大匹配数 = 最小点覆盖 = 总点数 - 最大独立集 = 总点数 - 最小路径覆盖
最小路径覆盖是选最少的边覆盖所有点
最小点覆盖是选最少的点覆盖所有边
原图的最大独立集就是补图的最大团
针对 DAG：原图的最小路径可重复覆盖 = 传递闭包后新图的最小路径覆盖
动态判断是否是二分图可以用带权并查集
树的最大匹配数和最大独立集一样，从叶子开始贪心

染色法判断二分图

```
int n, m;
int h[N], e[M], ne[M], w[M], idx;
int color[N]; // 0 表示未染色，1 表示染白色， 2 表示染黑色
void add(int a, int b, int c) {
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++;
}
bool dfs(int u, int c) {
    color[u] = c;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (!color[j] ) {
            if (!dfs(j, 3-c)) return false;
        }
        else if (color[j] == c) return false;
    }
    return true;
}
bool check() {
    memset(color, 0, sizeof color);
    bool flag = true;
    for (int i = 1; i <= n; i++) {
        if (!color[i]) {
            if (!dfs(i, 1)) {
                flag = false;
                break;
            }
        }
    }
    return flag;
}
```

匈牙利算法求二分图最大匹配 O(nm)

```
vector<int> adj[N]; // 邻接表存储所有边，匈牙利算法中只会用到从第一个集合指向第二个集合的边，所以这里只用存一个方向的边
int match[N];       // 存储第二个集合中的每个点当前匹配的第一个集合中的点是哪个
bool st[N];     // 表示第二个集合中的每个点是否已经被遍历过
```

```
bool find(int x) {
    for (auto v : adj[x]) {
        if (!st[v]) {
            st[v] = true;
            if (match[v] == 0 || find(match[v])) {
                match[v] = x;
                return true;
            }
        }
    }
    return false;
}
// 求最大匹配数，依次枚举第一个集合中的每个点能否匹配第二个集合中的点
int res = 0;
for (int i = 1; i <= n1; i++) {
    memset(st, false, sizeof(st));
    if (find(i)) res++;
}
```

连通性问题

有向图的强连通分量

有向图最少加 max（P，Q）条边成为强连通分量，P 和 Q 分别为缩点后入度为 0 的个数和出度为 0 的个数

缩点后 id 的顺序是拓扑排序的逆序

```
struct SCC {
    int n, cnt, timestamp, top;
    vector<vector<int>> g;
    vector<int> dfn, low, stk, id, sz, a, val;
    vector<bool> in_stk;
    SCC(int _n) {
        n = _n;
        g.resize(n + 1), dfn.resize(n + 1, 0), low.resize(n + 1, 0);
        stk.resize(n + 1, 0), id.resize(n + 1, 0), sz.resize(n + 1, 0);
        in_stk.resize(n + 1, 0);
        cnt = timestamp = top = 0;
    }
    void addEdge(int x, int y) {
        g[x].push_back(y);
    }
    void tarjan(int u) {
        dfn[u] = low[u] = ++timestamp;
        stk[++top] = u, in_stk[u] = 1;
        for (auto v : g[u]) {
            if (!dfn[v]) {
                tarjan(v);
                low[u] = min(low[u], low[v]);
            }
            else if (in_stk[v]) low[u] = min(low[u], dfn[v]);
        }
        if (dfn[u] == low[u]) {
            ++cnt;
            int y;
            do {
                y = stk[top--];
                in_stk[y] = 0;
                id[y] = cnt;
                sz[cnt]++;
            } while (y != u);
        }
    }
    void work() {
        for (int i = 1; i <= n; i++) {
            if (!dfn[i]) {
                tarjan(i);
            }
        }
    }
};
void solve() {
    int n;
    cin >> n;
    SCC scc(n);
    int m;
    cin >> m;
    for (int i = 1; i <= m; i++) {
        int a, b;
        cin >> a >> b;
```

```
        scc.addEdge(a, b);
    }
    scc.work();
    vector<vector<int>> adj(scc.cnt + 1);
    for (int i = 1; i <= n; i++) {
        for (auto v : scc.g[i]) {
            int a = scc.id[i], b = scc.id[v];
            if (a != b) {
                adj[a].push_back(b);
            }
        }
    }
}
```

最大半连通子图就是缩点后的最长链

```
int n, m, P;
int h[N], e[M], ne[M], idx, hs[N];
int dfn[N], low[N], timestamp;
int stk[N], top;
bool in_stk[N];
int id[N], scc_cnt, sz[N];
int dout[N], din[N];
int f[N], g[N]; // f 是最长链长，g 是方案数
void add(int h[], int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}
void tarjan(int u) {
    dfn[u] = low[u] = ++timestamp;
    stk[++top] = u, in_stk[u] = 1;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (!dfn[j]) {
            tarjan(j);
            low[u] = min(low[u], low[j]);
        }
        else if (in_stk[j]) low[u] = min(low[u], dfn[j]);
    }
    if (dfn[u] == low[u]) {
        ++scc_cnt;
        int y;
        do {
            y = stk[top--];
            in_stk[y] = 0;
            id[y] = scc_cnt;
            sz[scc_cnt]++;
        } while (y != u);
    }
}
void solve() {
    memset(h, -1, sizeof(h));
    memset(hs, -1, sizeof(hs));
    cin >> n >> m >> P;
    while (m--) {
        int a, b;
        cin >> a >> b;
        add(h, a, b);
    }
    for (int i = 1; i <= n; i++) {
        if (!dfn[i]) {
            tarjan(i);
        }
    }
    unordered_set<LL> S;
    for (int i = 1; i <= n; i++) {
        for (int j = h[i]; j != -1; j = ne[j]) {
            int k = e[j];
            int a = id[i], b = id[k];
            LL hashh = a * 100000011 + b;
            if (a != b && !S.count(hashh)) {
                add(hs, a, b);
                S.insert(hashh);
            }
        }
    }
    for (int i = scc_cnt; i >= 1; i--) {
        if (!f[i]) {
            f[i] = sz[i];
            g[i] = 1;
        }
```

```
        for (int j = hs[i]; j != -1; j = ne[j]) {
            int k = e[j];
            if (f[k] < f[i] + sz[k]) {
                f[k] = f[i] + sz[k];
                g[k] = g[i];
            }
            else if (f[k] == f[i] + sz[k]) g[k] = (g[k] + g[i]) % P;
        }
    }
    int maxf = 0, sum = 0;
    for (int i = 1; i <= n; i++) {
        if (f[i] > maxf) {
            maxf = f[i];
            sum = g[i];
        }
        else if (f[i] == maxf) sum = (sum + g[i]) % P;
    }
    cout << maxf << '\n' << sum << '\n';
}
```

边双连通分量

在一张连通的无向图中，对于两个点 u 和 v，如果无论删去哪条边（只能删去一条）都不能使它们不连通，我们就说 u 和 v 边双连通。

无向图最少加 (cnt + 1) / 2 条边成为边双连通分量 cnt 为缩点后树上度数为 1 的点的个数

```
int n, m;
int h[N], e[M], ne[M], idx;
int dfn[N], low[N], timestamp;
int stk[N], top;
int id[N], dcc_cnt;
int d[N]; // 缩点后的度数
bool is_bridge[M];
vector<int> point[N]; // 记录每个连通分量有哪些点
void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}
void tarjan(int u, int from) {
    dfn[u] = low[u] = ++timestamp;
    stk[++top] = u;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (!dfn[j]) {
            tarjan(j, i);
            low[u] = min(low[u], low[j]);
            if (dfn[u] < low[j]) is_bridge[i] = is_bridge[i ^ 1] = 1;
        }
        else if (i != (from ^ 1)) low[u] = min(low[u], dfn[j]);
    }
    if (dfn[u] == low[u]) {
        ++dcc_cnt;
        int y;
        do {
            y = stk[top--];
            id[y] = dcc_cnt;
        } while (y != u);
    }
}
void solve() {
    cin >> n >> m;
    memset(h, -1, sizeof(h));
    while (m--) {
        int a, b;
        cin >> a >> b;
        add(a, b), add(b, a);
    }
    for (int i = 1; i <= n; i++) {
        if (!dfn[i]) {
            tarjan(i, -1);
        }
    }
    for (int i = 1; i <= n; i++) {
        int p = id[i];
        point[p].push_back(i);
    }
    for (int i = 0; i < idx; i++) {
        if (is_bridge[i]) {
            d[id[e[i]]]++;
        }
    }
```

```
}
```

点双连通分量

在一张连通的无向图中，对于两个点 u 和 v，如果无论删去哪个点（只能删去一个，且不能删 u 和 v 自己）都不能使它们不连通，我们就说 u 和 v 点双连通。

一个点可能属于多个点双，但是一条边属于恰好一个点双

```
int n, m;
int h[N], e[M], ne[M], idx;
int dfn[N], low[N], timestamp;
int stk[N], top;
int dcc_cnt;
vector<int> point[N];
bool is_cut_point[N];
int root;
void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}
void tarjan(int u) {
    dfn[u] = low[u] = ++timestamp;
    stk[++top] = u;
    if (u == root && h[u] == -1) {
        dcc_cnt++;
        point[dcc_cnt].push_back(u);
        return;
    }
    int cnt = 0;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (!dfn[j]) {
            tarjan(j);
            low[u] = min(low[u], low[j]);
            if (dfn[u] <= low[j]) {
                cnt++;
                if (u != root || cnt > 1) is_cut_point[u] = 1;
                ++dcc_cnt;
                int y;
                do {
                    y = stk[top--];
                    point[dcc_cnt].push_back(y);
                } while (y != j);
                point[dcc_cnt].push_back(u);
            }
        }
        else low[u] = min(low[u], dfn[j]);
    }
}
void solve() {
    cin >> n >> m;
    memset(h, -1, sizeof(h));
    while (m--) {
        int a, b;
        cin >> a >> b;
        if (a == b) continue;
        add(a, b), add(b, a);
    }
    for (int i = 1; i <= n; i++) {
        if (!dfn[i]) {
            root = i;
            tarjan(i);
        }
    }
}
```

圆方树

记得点开两倍

原来的每个点对应一个圆点，每一个点双对应一个方点。所以共有 n + c 个点，其中 n 是原图点数，c 是原图点双连通分量的个数。对于每一个点双连通分量，它对应的方点向这个点双连通分量中的每个点连边。每个点双形成一个「菊花图」，多个「菊花图」通过原图中的割点连接在一起（因为点双的分隔点是割点）。

```
int n, m, cnt;
vector<int> G[N], t[N * 2];
int dfn[N], low[N], dfc;
int stk[N], tp;
int sz[N * 2], w[N * 2]; // 树上子树大小和树上点权值
```

```
void Tarjan(int u) {
    low[u] = dfn[u] = ++dfc;                // low 初始化为当前节点 dfn
    stk[++tp] = u;                          // 加入栈中
    for (int v : G[u]) {                    // 遍历 u 的相邻节点
        if (!dfn[v]) {                        // 如果未访问过
            Tarjan(v);                          // 递归
            low[u] = min(low[u], low[v]);  // 未访问的和 low 取 min
            if (low[v] == dfn[u]) {    // 标志着找到一个以 u 为根的点双连通分量
                ++cnt;                 // 增加方点个数
                // 将点双中除了 u 的点退栈，并在圆方树中连边
                for (int x = 0; x != v; --tp) {
                    x = stk[tp];
                    T[cnt].push_back(x);
                    T[x].push_back(cnt);
                }
                // 注意 u 自身也要连边（但不退栈）
                T[cnt].push_back(u);
                T[u].push_back(cnt);
            }
        }
        else low[u] = min(low[u], dfn[v]);  // 已访问的和 dfn 取 min
    }
}
void solve() {
    cin >> n >> m;
    cnt = n;
    for (int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    for (int i = 1; i <= n; i++) {
        if (!dfn[i]) {
            Tarjan(i);
            --tp;
        }
    }
}
```

欧拉回路和路径

全连通无向图欧拉路径（不回路）要求只有两个点是奇数度

全连通无向图欧拉回路没有奇数度的点

全连通有向图欧拉路径：起点出度比入度多一，终点入度比出度多一，其余点入度等于出度

全连通有向图欧拉回路：所有点入度等于出度

欧拉回路

```
int type; // 1为无向，2为有向
int n, m;
int h[N], e[M], ne[M], idx;
bool used[M]; // 判断边是否被用过
int ans[M], cnt;  // ans记录的是第几个边，值为正是正向边，负是反向边
int din[N], dout[N];
void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}
void dfs(int u) {
    for (int &i = h[u]; i != -1;) {
        if (used[i]) {
            i = ne[i];
            continue;
        }
        used[i] = 1;
        if (type == 1) used[i ^ 1] = 1;
        int t;
        if (type == 1) { // 无向边加了两边，有向边只有一遍
            t = i / 2 + 1;
            if (i & 1) t *= -1;
        }
        else t = i + 1;
        int j = e[i];
        i = ne[i];
        dfs(j);
        ans[++cnt] = t;
    }
```

```
}
void solve() {
    cin >> type;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) h[i] = -1;
    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        add(a, b);
        if (type == 1) add(b, a);
        din[b]++, dout[a]++;
    }
    if (type == 1) {
        for (int i = 1; i <= n; i++) {
            if ((din[i] + dout[i]) & 1) {
                cout << "NO" << '\n';
                return;
            }
        }
    }
    else {
        for (int i = 1; i <= n; i++) {
            if (din[i] != dout[i]) {
                cout << "NO" << '\n';
                return;
            }
        }
    }
    for (int i = 1; i <= n; i++) { // 从任意一个有边的起点开始搜
        if (h[i] != -1) {
            dfs(i);
            break;
        }
    }
    if (cnt < m) {
        cout << "NO" << '\n';
        return;
    }
    cout << "YES" << '\n';
    for (int i = cnt; i >= 1; i--) cout << ans[i] << ' ';
}
```

欧拉路径

```
int n = 500, m;
int g[N][N];
int ans[1100], cnt;
int d[N];
void dfs(int u) {
    for (int i = 1; i <= n; i++) { // 字典序最小
        if (g[u][i]) {
            g[u][i]--, g[i][u]--;
            dfs(i);
        }
    }
    ans[++cnt] = u;
}
void solve() {
    cin >> m;
    while (m--) {
        int a, b;
        cin >> a >> b;
        g[a][b]++, g[b][a]++;
        d[a]++, d[b]++;
    }
    int start = 1;
    while (!d[start]) start++;
    for (int i = 1; i <= n; i++) {
        if (d[i] & 1) {
            start = i;
            break;
        }
    }
    dfs(start);
    for (int i = cnt; i >= 1; i--) cout << ans[i] << '\n';
}
```

全局最小割

有无向图 G = (V, E)，设 C 为图 G 中一些弧的集合，若从 G 中删去 C 中的所有弧能使图 G 不是连通图，称 C 图 G 的一个割。

全局最小割：包含的弧的权和最小的割，称为全局最小割。 O(VE + V^2logV)

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 610, INF = 1e9;
int n, m, x, y, z, s, t, dis[MAXN][MAXN], w[MAXN], dap[MAXN], vis[MAXN], ord[MAXN];
int proc (int x) {
  memset(vis, 0, sizeof(vis));
  memset(w, 0, sizeof(w));
  w[0] = -1;
  for (int i = 1; i <= n - x + 1; i++) {
    int mx = 0;
    for (int j = 1; j <= n; j++) {
      if (!dap[j] && !vis[j] && w[j] > w[mx]) {
        mx = j;
      }
    }
    vis[mx] = 1, ord[i] = mx;
    for (int j = 1; j <= n; j++) {
      if (!dap[j] && !vis[j]) {
        w[j] += dis[mx][j];
      }
    }
  }
  s = ord[n - x], t = ord[n - x + 1];
  return w[t];
}
int sw () {
  int res = INF;
  for (int i = 1; i < n; i++) {
    res = min(res, proc(i));
    dap[t] = 1;
    for (int j = 1; j <= n; j++) {
      dis[s][j] += dis[t][j];
      dis[j][s] += dis[j][t];
    }
  }
  return res;
}
int main () {
  cin >> n >> m;
  for (int i = 1; i <= m; i++) {
    cin >> x >> y >> z;
    dis[x][y] += z, dis[y][x] += z;
  }
  cout << sw();
  return 0;
}
```

树上K级祖先

每次询问一个点的 k 级祖先是谁, 长链剖分 复杂度O(nlogn + q)

```
#include<bits/stdc++.h>
#define REG register
#define LL long long
#define UI unsigned int
#define MAXN 500005
using namespace std;
inline int read() {
    REG int x(0);
    REG char c = getchar();
    while (!isdigit(c)) c = getchar();
    while (isdigit(c)) x = (x * 10) + (c ^ 48), c = getchar();
    return x;
}
int n, q, rt;
vector<int> NodeUp[MAXN], NodeDown[MAXN], Edge[MAXN];
int Dep[MAXN], MDep[MAXN], Son[MAXN], Top[MAXN], HighBit[MAXN];
int Fat[MAXN][21];
LL ans;
int lastans;
UI s;
UI Get(UI x) {
    x ^= x << 13;
    x ^= x >> 17;
    x ^= x << 5;
    return s = x;
```

```
}
void dfs1(int now) {
    MDep[now] = Dep[now] = Dep[Fat[now][0]] + 1;
    for (auto v : Edge[now]) {
        Fat[v][0] = now;
        for (REG int i = 0; Fat[v][i]; i++) Fat[v][i + 1] = Fat[Fat[v][i]][i];
        dfs1(v);
        if (MDep[v] > MDep[now]) MDep[now] = MDep[v], Son[now] = v;
    }
}
void dfs2(int now, int top) {
    Top[now] = top;
    if (now == top) {
        for (REG int i = 0, f = now; i <= MDep[now] - Dep[now]; i++)
            NodeUp[now].push_back(f), f = Fat[f][0];
        for (REG int i = 0, f = now; i <= MDep[now] - Dep[now]; i++)
            NodeDown[now].push_back(f), f = Son[f];
    }
    if (Son[now]) dfs2(Son[now], top);
    for (auto v : Edge[now])
        if (v ^ Son[now]) dfs2(v, v);
}
inline int Ask(int x, int k) {
    if (!k) return x;
    x = Fat[x][HighBit[k]], k -= (1 << HighBit[k]), k -= Dep[x] - Dep[Top[x]], x = Top[x];
    return k >= 0 ? NodeUp[x][k] : NodeDown[x][-k];
}
void Solve() {
    n = read(), q = read(), s = read(), HighBit[1] = 0;
    for (REG int i = 2; i <= n; i++)
        HighBit[i] = HighBit[i >> 1] + 1;
    for (REG int i = 1; i <= n; i++)
        Edge[read()].push_back(i);
    rt = Edge[0][0];
    dfs1(rt);
    dfs2(rt, rt);
    for (REG int i = 1; i <= q; i++) {
        int x = ((Get(s) ^ lastans) % n) + 1;
        int k = ((Get(s) ^ lastans) % Dep[x]);
        lastans = Ask(x, k);
        ans ^= 1ll * i * lastans;
    }
    printf("%lld\n", ans);
}
int main() {
    Solve();
    return 0;
}
```

斯坦纳树

最小斯坦纳树允许在给定点外增加额外的点，使生成的最短网络开销最小。

```
const int INF = 0x3f3f3f3f;
void solve() {
    int n, m, k;
    cin >> n >> m >> k;
    vector<vector<pair<int, int>>> g(n + 1);
    vector<bool> vis(n + 1, 0);
    vector<vector<int>> dp((1 << k) + 1, vector<int> (n + 1, INF));
    auto djs = [&](int S) -> void {
        priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> q;
        for (int i = 1; i <= n; i++) vis[i] = 0;
        for (int i = 1; i <= n; i++) {
            if (dp[S][i] != INF) {
                q.push({dp[S][i], i});
            }
        }
        while (!q.empty()) {
            auto u = q.top();
            q.pop();
            if (vis[u.second]) continue;
            vis[u.second] = 1;
            for (auto [v, w] : g[u.second]) {
                if (dp[S][v] > u.first + w) {
                  dp[S][v] = u.first + w;
                  q.push({dp[S][v], v});
                }
            }
        }
```

```
    };
    for (int i = 1; i <= m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        g[u].emplace_back(v, w), g[v].emplace_back(u, w);
    }
    for (int i = 0; i < k; i++) {
        int x;
        cin >> x;
        dp[1 << i][x] = 0;
    }
    for (int S = 1; S < (1 << k); S++) {
        for (int T = S & (S - 1); T; T = S & (T - 1)) {
            if (T < (S ^ T)) break;
            for (int i = 1; i <= n; i++) dp[S][i] = min(dp[S][i], dp[T][i] + dp[T ^ S][i]);
        }
        djs(S);
    }
    int ans = INF;
    for (auto u : dp[(1 << k) - 1]) ans = min(ans, u);
    cout << ans << '\n';
}
```

稳定婚姻系统

为两个元素数量相等的集合寻找稳定匹配，确保不存在彼此更偏好的未配对组合

```
#include <bits/stdc++.h>
using namespace std;
mt19937_64 rnd(chrono::steady_clock::now().time_since_epoch().count());
using LL = long long;
const int MAXN = 1e3 + 10;
// pref[i][j]表示女性 i 对男性 j 的偏好程度。
// order[i][j]表示男性 i 对女性 j 的偏好排序。
// nex 表示当前每个男性的提议顺序，记录下一个要向哪个女性求婚。
int pref[MAXN][MAXN], order[MAXN][MAXN], nex[MAXN];
int future_husband[MAXN], future_wife[MAXN];
// 队列用于存储尚未配对的男性。
queue<int> q;
void engage(int man, int woman) {
    int m = future_husband[woman];
    if (m) {
        future_wife[m] = 0;
        q.push(m);
    }
    future_wife[man] = woman;
    future_husband[woman] = man;
}
void solve() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        int which;
        cin >> which;
        for (int j = 1; j <= n; j++) {
            int x;
            cin >> x;
            order[which][x] = j;
        }
        future_husband[which] = 0;
    }
    for (int i = 1; i <= n; i++) {
        int which;
        cin >> which;
        for (int j = 1; j <= n; j++) cin >> pref[which][j];
        nex[which] = 1;
        future_wife[which] = 0;
        q.push(which);
    }
    while (!q.empty()) {
        int man = q.front();
        q.pop();
        int woman = pref[man][nex[man]++];
        if (!future_husband[woman]) engage(man, woman);
        else if (order[woman][man] < order[woman][future_husband[woman]]) engage(man, woman);
        else q.push(man);
    }
    while (!q.empty()) q.pop();
    for (int i = 1; i <= n; i++) cout << i << " " << future_wife[i] << '\n';
}
```

```
signed main() {
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    int T = 1;
    cin >> T;
    while (T--) solve();
    return 0;
}
```

无向图三元环计数

O(m * sqrt(m))

```
int n, m;
int h[N], e[M], ne[M], idx;
int deg[N], vis[N], ans;
struct Edge {
    int u, v;
} edge[M];
void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}
void solve() {
    cin >> n >> m;
    memset(h, -1, sizeof(h));
    for (int i = 1; i <= m; i++) {
        int a, b;
        cin >> a >> b;
        edge[i] = {a, b};
        deg[a]++, deg[b]++;
    }
    for (int i = 1; i <= m; i++) {
        int &u = edge[i].u, &v = edge[i].v;
        if (deg[u] < deg[v] || (deg[u] == deg[v] && u > v)) swap(u, v);
        add(u, v);
    }
    for (int x = 1; x <= n; x++) {
        for (int i = h[x]; i != -1; i = ne[i]) vis[e[i]] = x;
        for (int i = h[x]; i != -1; i = ne[i]) {
            int y = e[i];
            for (int j = h[y]; j != -1; j = ne[j]) {
                int z = e[j];
                if (vis[z] == x) ans++;
            }
        }
    }
    cout << ans << '\n';
}
```

虚树

```
const int N = 2e5 + 10, LOG = 20;
vector<int> g[N];
int up[N][LOG];
int depthv[N], tin[N], tout[N], timer;
void dfs_root(int u, int fa) {
    tin[u] = ++timer;
    up[u][0] = fa;
    for (int i = 1; i < LOG; i++) up[u][i] = up[up[u][i - 1]][i - 1];
    for (auto v : g[u]) {
        if (v == fa) continue;
        depthv[v] = depthv[u] + 1;
        dfs_root(v, u);
    }
    tout[u] = timer;
}
bool is_ancestor(int u, int v) {
    return tin[u] <= tin[v] && tout[v] <= tout[u];
}
int lca(int u, int v) {
    if (is_ancestor(u, v)) return u;
    if (is_ancestor(v, u)) return v;
    for (int i = LOG - 1; i >= 0; i--) {
        if (!is_ancestor(up[u][i], v)) {
            u = up[u][i];
        }
    }
    return up[u][0];
}
```

```
pair<vector<bool>, int> build(const vector<int>& nodes, vector<int>& vt_nodes, vector<vector<int>>& vt_adj) {
    if (nodes.empty()) {
        vt_nodes.clear();
        vt_adj.clear();
        return {{}, -1};
    }
    // 1) 去重 + 按 tin 排序
    vector<int> a = nodes;
    sort(a.begin(), a.end(), [&](int x, int y) {
        return tin[x] < tin[y];
    });
    a.erase(unique(a.begin(), a.end()), a.end());
    // 2) 加相邻 LCA
    vt_nodes = a;
    vt_nodes.reserve(a.size() * 2);
    for (int i = 0; i + 1 < (int)a.size(); i++) vt_nodes.push_back(lca(a[i], a[i + 1]));
    // 3) 最终集合（按 tin 排序 + 去重）
    sort(vt_nodes.begin(), vt_nodes.end(), [&](int x, int y) {
        return tin[x] < tin[y];
    });
    vt_nodes.erase(unique(vt_nodes.begin(), vt_nodes.end()), vt_nodes.end());
    // 4) 用标准"栈法"连边（父→子），边先用原节点编号存
    vector<pair<int, int>> edges; // (parent_node, child_node)
    vector<int> st;
    auto link = [&](int p, int ch) -> void {
        edges.emplace_back(p, ch);
    };
    st.push_back(vt_nodes[0]);
    for (int i = 1; i < (int)vt_nodes.size(); i++) {
        int u = vt_nodes[i];
        int L = lca(u, st.back());
        if (L == st.back()) {                // 直接挂在当前链上
            st.push_back(u);
            continue;
        }
        // 把栈弹到 L 之下
        while ((int)st.size() >= 2 && tin[st[st.size() - 2]] >= tin[L]) {
            link(st[st.size() - 2], st.back());
            st.pop_back();
        }
        if (st.back() != L) {                // 把 L 接到当前链上
            link(L, st.back());
            st.pop_back();
            st.push_back(L);
        }
        st.push_back(u);
    }
    while ((int)st.size() > 1) {             // 清栈，连剩余边
        link(st[st.size() - 2], st.back());
        st.pop_back();
    }
    int root_node = st.back();               // 栈最后剩下的就是虚树根(原树编号)
    // 5) 把"原节点编号的边"映射到 vt_nodes 的下标，形成 vt_adj
    map<int, int> idx;
    for (int i = 0; i < (int)vt_nodes.size(); i++) idx[vt_nodes[i]] = i;
    vt_adj.assign(vt_nodes.size(), {});
    for (auto [p, ch] : edges) {
        int ip = idx[p], ic = idx[ch];
        vt_adj[ip].push_back(ic);
    }
    // 6) vis 标记哪些是原始 nodes
    vector<bool> vis(vt_nodes.size(), false);
    for (auto x : a) vis[idx[x]] = true;
    int root_idx = idx[root_node];
    return {vis, root_idx};
}
void solve() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) g[i].clear();
    for (int i = 1; i <= n - 1; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    timer = 0;
    depthv[1] = 0;
    for (int i = 0; i < LOG; i++) up[1][i] = 1;
    dfs_root(1, 1);
    int q;
```

```
    cin >> q;
    while (q--) {
        int k;
        cin >> k;
        vector<int> nodes, vt_nodes;
        vector<vector<int>> vt_adj;
        for (int i = 1; i <= k; i++) {
            int x;
            cin >> x;
            nodes.push_back(x);
        }
        // 判断无解最好先判，不用建虚树
        auto [vis, root] = build(nodes, vt_nodes, vt_adj);
        // 后面可在虚树上dp
    }
}
```

严格次小生成树

树上倍增

```
int n, m;
int h[N], e[M], w[M], ne[M], idx;
struct Edge {
    int a, b, w;
    bool used;
    bool operator< (const Edge &W) const {
        return w < W.w;
    }
} edge[M];
int p[N], ra[N];
int depth[N], fa[N][17], d1[N][17], d2[N][17];
void add(int a, int b, int c) {
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++;
}
int find(int x) {
    if (p[x] != x) p[x] = find(p[x]);
    return p[x];
}
LL kru() {
    for (int i = 1; i <= n; i++) p[i] = i;
    sort(edge, edge + m);
    LL res = 0;
    for (int i = 0; i < m; i++) {
        int a = edge[i].a, b = edge[i].b, w = edge[i].w;
        a = find(a), b = find(b);
        if (a != b) {
            if (ra[a] <= ra[b]) {
                p[a] = b;
                if (ra[a] == ra[b]) ra[b]++;
            }
            else p[b] = a;
            res += w;
            edge[i].used = 1;
        }
    }
    return res;
}
void build() {
    memset(h, -1, sizeof(h));
    for (int i = 0; i < m; i++) {
        if (edge[i].used) {
            int a = edge[i].a, b = edge[i].b, w = edge[i].w;
            add(a, b, w), add(b, a, w);
        }
    }
}
void bfs() {
    memset(depth, 0x3f, sizeof(depth));
    depth[0] = 0, depth[1] = 1;
    queue<int> q;
    q.push(1);
    while (q.size()) {
        auto k = q.front();
        q.pop();
        for (int i = h[k]; i != -1; i = ne[i]) {
            int j = e[i];
            if (depth[j] > depth[k] + 1) {
                depth[j] = depth[k] + 1;
                q.push(j);
```

```
                fa[j][0] = k;
                d1[j][0] = w[i], d2[j][0] = -INF;
                for (int t = 1; t <= 16; t++) {
                    int anc = fa[j][t - 1];
                    fa[j][t] = fa[anc][t - 1];
                    int distance[4] = {d1[j][t - 1], d2[j][t - 1], d1[anc][t - 1], d2[anc][t - 1]};
                    for (int u = 0; u < 4; u++) {
                        int d = distance[u];
                        if (d > d1[j][t]) d2[j][t] = d1[j][t], d1[j][t] = d;
                        else if (d != d1[j][t] && d > d2[j][t]) d2[j][t] = d;
                    }
                }
            }
        }
    }
}
int lca(int a, int b, int w) {
    static int distance[N * 2];
    int cnt = 0;
    if (depth[a] < depth[b]) swap(a, b);
    for (int k = 16; k >= 0; k--) {
        if (depth[fa[a][k]] >= depth[b]) {
            distance[cnt++] = d1[a][k];
            distance[cnt++] = d2[a][k];
            a = fa[a][k];
        }
    }
    if (a != b) {
        for (int k = 16; k >= 0; k--) {
            if (fa[a][k] != fa[b][k]) {
                distance[cnt++] = d1[a][k];
                distance[cnt++] = d2[a][k];
                distance[cnt++] = d1[b][k];
                distance[cnt++] = d2[b][k];
                a = fa[a][k], b = fa[b][k];
            }
        }
        distance[cnt++] = d1[a][0];
        distance[cnt++] = d1[b][0];
    }
    int dist1 = -INF, dist2 = -INF;
    for (int i = 0; i < cnt; i++) {
        int d = distance[i];
        if (d > dist1) dist2 = dist1, dist1 = d;
        else if (d != dist1 && d > dist2) dist2 = d;
    }
    if (w > dist1) return w - dist1;
    if (w > dist2) return w - dist2;
    return INF;
}
void solve() {
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        edge[i] = {a, b, c};
    }
    LL sum = kru();
    build();
    bfs();
    LL res = 1e18;
    for (int i = 0; i < m; i++) {
        if (!edge[i].used) {
            int a = edge[i].a, b = edge[i].b, w = edge[i].w;
            res = min(res, sum + lca(a, b, w));
        }
    }
    cout << res << '\n';
}
```

一般图匹配

一般图最大匹配 O(n^3)

```
const int N = 1010;
int n, m;
vector<int> g[N];
int match[N], p[N], baseArr[N];
bool used[N], blossom[N];
int lca(int a, int b) {
```

```
    vector<bool> usedVertex(n + 1, false);
    while (true) {
        a = baseArr[a];
        usedVertex[a] = true;
        if (match[a] == -1) break;
        a = p[match[a]];
    }
    while (true) {
        b = baseArr[b];
        if (usedVertex[b]) return b;
        b = p[match[b]];
    }
    return -1;
}
void markPath(int v, int b, int x) {
    while (baseArr[v] != b) {
        blossom[baseArr[v]] = blossom[baseArr[match[v]]] = true;
        p[v] = x;
        x = match[v];
        v = p[match[v]];
    }
}
bool findPath(int start) {
    for (int i = 1; i <= n; i++) {
        used[i] = false;
        p[i] = -1;
        baseArr[i] = i;
    }
    queue<int> q;
    q.push(start);
    used[start] = true;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (int u : g[v]) {
            if (baseArr[v] == baseArr[u] || match[v] == u) continue;
            if (u == start || (match[u] != -1 && p[match[u]] != -1)) {
                int cur = lca(v, u);
                for (int i = 1; i <= n; i++) blossom[i] = false;
                markPath(v, cur, u);
                markPath(u, cur, v);
                for (int i = 1; i <= n; i++) {
                    if (blossom[baseArr[i]]) {
                        baseArr[i] = cur;
                        if (!used[i]) {
                            used[i] = true;
                            q.push(i);
                        }
                    }
                }
            }
            else if (p[u] == -1) {
                p[u] = v;
                if (match[u] == -1) {
                    int cur = u;
                    while (cur != -1) {
                        int pv = p[cur];
                        int nxt = match[pv];
                        match[cur] = pv;
                        match[pv] = cur;
                        cur = nxt;
                    }
                    return true;
                }
                else {
                    int nxt = match[u];
                    used[nxt] = true;
                    q.push(nxt);
                }
            }
        }
    }
    return false;
}
int edmonds() {
    for (int i = 1; i <= n; i++) match[i] = -1;
    int matchingSize = 0;
    for (int i = 1; i <= n; i++) {
        if (match[i] == -1)
            if (findPath(i))
                matchingSize++;
```

```
    }
    return matchingSize;
}
void solve() {
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    int ans = edmonds();
    cout << ans << "\n";
    for (int i = 1; i <= n; i++) {
        if (match[i] != -1) cout << match[i] << ' ';
        else cout << 0 << ' ';
    }
}
```

一般图最大权匹配 O(n^3)

```
const int N = 620, INF = 0x3f3f3f3f; // 点数设置为 1.5 倍的点数
struct Edge {
    int x, y, z;
    Edge() : x(0), y(0), z(0) {}
    Edge(int a, int b, int c) : x(a), y(b), z(c) {}
};
Edge g[N][N];
int n, m, nx, t;
int lab[N], match[N], slack[N];
int st[N], pa[N], flower_from[N][N], S[N], vis[N];
vector<int> flower[N];
deque<int> q;
int dist(const Edge &e) {
    return lab[e.x] + lab[e.y] - e.z * 2;
}
void update_slack(int x, int y) {
    if (!slack[y] || dist(g[x][y]) < dist(g[slack[y]][y])) slack[y] = x;
}
void set_slack(int y) {
    slack[y] = 0;
    for (int x = 1; x <= n; x++) {
        if (g[x][y].z > 0 && st[x] != y && S[st[x]] == 0) {
            update_slack(x, y);
        }
    }
}
void q_push(int x) {
    if (x <= n) q.push_back(x);
    else {
        for (int i = 0; i < flower[x].size(); i++) {
            q_push(flower[x][i]);
        }
    }
}
void set_st(int x, int b) {
    st[x] = b;
    if (x <= n) return;
    for (int i = 0; i < flower[x].size(); i++) set_st(flower[x][i], b);
}
int get_pr(int b, int xr) {
    int pr = find(flower[b].begin(), flower[b].end(), xr) - flower[b].begin();
    if (pr % 2 == 1) {
        reverse(flower[b].begin() + 1, flower[b].end());
        return flower[b].size() - pr;
    }
    return pr;
}
void set_match(int x, int y) {
    match[x] = g[x][y].y;
    if (x <= n) return;
    Edge e = g[x][y];
    int xr = flower_from[x][e.x], pr = get_pr(x, xr);
    for (int i = 0; i < pr; ++i) set_match(flower[x][i], flower[x][i ^ 1]);
    set_match(xr, y);
    rotate(flower[x].begin(), flower[x].begin() + pr, flower[x].end());
}
void augment(int x, int y) {
    int xnv = st[match[x]];
    set_match(x, y);
```

```
    if (!xnv) return;
    set_match(xnv, st[pa[xnv]]);
    augment(st[pa[xnv]], xnv);
}
int get_lca(int x, int y) {
    for (++t; x || y; swap(x, y)) {
        if (x == 0) continue;
        if (vis[x] == t) return x;
        vis[x] = t;
        x = st[match[x]];
        if (x) x = st[pa[x]];
    }
    return 0;
}
void add_blossom(int x, int lca, int y) {
    int b = n + 1;
    while (b <= nx && st[b]) b++;
    if (b > nx) nx++;
    lab[b] = 0;
    S[b] = 0;
    match[b] = match[lca];
    flower[b].clear();
    flower[b].push_back(lca);
    for (int xx = x, yy; xx != lca; xx = st[pa[yy]]) {
        flower[b].push_back(xx);
        flower[b].push_back(yy = st[match[xx]]);
        q_push(yy);
    }
    reverse(flower[b].begin() + 1, flower[b].end());
    for (int xx = y, yy; xx != lca; xx = st[pa[yy]]) {
        flower[b].push_back(xx);
        flower[b].push_back(yy = st[match[xx]]);
        q_push(yy);
    }
    set_st(b, b);
    for (int xx = 1; xx <= nx; ++xx) g[b][xx].z = g[xx][b].z = 0;
    for (int xx = 1; xx <= n; ++xx) flower_from[b][xx] = 0;
    for (int i = 0; i < flower[b].size(); i++) {
        int xs = flower[b][i];
        for (int xx = 1; xx <= nx; xx++) {
            if (g[b][xx].z == 0 || dist(g[xs][xx]) < dist(g[b][xx])) {
                g[b][xx] = g[xs][xx];
                g[xx][b] = g[xx][xs];
            }
        }
        for (int xx = 1; xx <= n; xx++) {
            if (flower_from[xs][xx]) {
                flower_from[b][xx] = xs;
            }
        }
    }
    set_slack(b);
}
void expand_blossom(int b) {
    for (int i = 0; i < flower[b].size(); i++) set_st(flower[b][i], flower[b][i]);
    int xr = flower_from[b][g[b][pa[b]].x], pr = get_pr(b, xr);
    for (int i = 0; i < pr; i += 2) {
        int xs = flower[b][i], xns = flower[b][i + 1];
        pa[xs] = g[xns][xs].x;
        S[xs] = 1;
        S[xns] = 0;
        slack[xs] = 0;
        set_slack(xns);
        q_push(xns);
    }
    S[xr] = 1;
    pa[xr] = pa[b];
    for (int i = pr + 1; i < flower[b].size(); i++) {
        int xs = flower[b][i];
        S[xs] = -1;
        set_slack(xs);
    }
    st[b] = 0;
}
bool on_found_edge(const Edge &e) {
    int x = st[e.x], y = st[e.y];
    if (S[y] == -1) {
        pa[y] = e.x;
        S[y] = 1;
        int nu = st[match[y]];
        slack[y] = slack[nu] = 0;
```

```
        S[nu] = 0;
        q_push(nu);
    }
    else if (S[y] == 0) {
        int lca = get_lca(x, y);
        if (!lca) {
            augment(x, y);
            augment(y, x);
            return true;
        }
        else add_blossom(x, lca, y);
    }
    return false;
}
bool matching() {
    memset(S, -1, sizeof(S));
    memset(slack, 0, sizeof(slack));
    q.clear();
    for (int x = 1; x <= nx; x++) {
        if (st[x] == x && !match[x]) {
            pa[x] = 0;
            S[x] = 0;
            q_push(x);
        }
    }
    if (q.empty()) return false;
    while (true) {
        while (q.size()) {
            int x = q.front();
            q.pop_front();
            if (S[st[x]] == 1) continue;
            for (int y = 1; y <= n; y++) {
                if (g[x][y].z > 0 && st[x] != st[y]) {
                    if (dist(g[x][y]) == 0) {
                        if (on_found_edge(g[x][y])) return true;
                    }
                    else update_slack(x, st[y]);
                }
            }
        }
        int d = INF;
        for (int b = n + 1; b <= nx; b++) {
            if (st[b] == b && S[b] == 1) {
                d = min(d, lab[b] / 2);
            }
        }
        for (int x = 1; x <= nx; x++) {
            if (st[x] == x && slack[x]) {
                if (S[x] == -1) d = min(d, dist(g[slack[x]][x]));
                else if (S[x] == 0) d = min(d, dist(g[slack[x]][x]) / 2);
            }
        }
        for (int x = 1; x <= n; x++) {
            if (S[st[x]] == 0) {
                if (lab[x] <= d) return false;
                lab[x] -= d;
            }
            else if (S[st[x]] == 1) lab[x] += d;
        }
        for (int b = n + 1; b <= nx; b++) {
            if (st[b] == b) {
                if (S[st[b]] == 0) lab[b] += d * 2;
                else if (S[st[b]] == 1) lab[b] -= d * 2;
            }
        }
        q.clear();
        for (int x = 1; x <= nx; x++) {
            if (st[x] == x && slack[x] && st[slack[x]] != x && dist(g[slack[x]][x]) == 0) {
                if (on_found_edge(g[slack[x]][x])) return true;
            }
        }
        for (int b = n + 1; b <= nx; b++) {
            if (st[b] == b && S[b] == 1 && lab[b] == 0) {
                expand_blossom(b);
            }
        }
    }
    return false;
}
pair<LL, int> weight_blossom() {
    memset(match, 0, sizeof(match));
```

```
    nx = n;
    int n_matches = 0;
    LL tot_weight = 0;
    for (int x = 0; x <= n; x++) {
        st[x] = x;
        flower[x].clear();
    }
    int w_max = 0;
    for (int x = 1; x <= n; x++) {
        for (int y = 1; y <= n; y++) {
            flower_from[x][y] = (x == y ? x : 0);
            w_max = max(w_max, g[x][y].z);
        }
    }
    for (int x = 1; x <= n; x++) lab[x] = w_max;

    while (matching()) ++n_matches;

    for (int x = 1; x <= n; x++) {
        if (match[x] && match[x] < x) {
            tot_weight += (LL)g[x][match[x]].z;
        }
    }
    return make_pair(tot_weight, n_matches);
}
void solve() {
    cin >> n >> m;
    for (int x = 1; x <= n; x++) {
        for (int y = 1; y <= n; y++) {
            g[x][y] = Edge(x, y, 0);
        }
    }
    int x, y, z;
    for (int i = 1; i <= m; i++) {
        cin >> x >> y >> z;
        g[x][y].z = g[y][x].z = z;
    }
    cout << weight_blossom().first << "\n";
    for (int i = 1; i <= n; i++) cout << match[i] << ' ';
    cout << '\n';
}
```

最短哈密顿路径

状态压缩dp 从 0 号点到 n - 1 号点的最短哈密顿路径

```
const int N = 20, M = 1 << 20, mod = 1e9 + 7;
int g[N][N];
int f[M][N];
void solve() {
    memset(f, 0x3f, sizeof(f));
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> g[i][j];
        }
    }
    f[1][0] = 0;
    for (int i = 0; i < 1 << n; i++) {
        for (int j = 0; j < n; j++) {
            if (i >> j & 1) {
                for (int k = 0; k < n; k++) {
                    if (i >> k & 1) {
                        f[i][j] = min(f[i][j], f[i - (1 << j)][k] + g[k][j]);
                    }
                }
            }
        }
    }
    cout << f[(1 << n) - 1][n - 1];
}
```

最短路和次短路及方案数

正权图

```
struct Ver {
    int id, type, dist; // type0 为最短路， type1 为次短路
```

```
    bool operator> (const Ver &w) const {
        return dist > w.dist;
    }
};
int n, m, S, T;
int h[N], e[M], w[M], ne[M], idx;
int dist[N][2], cnt[N][2];
bool vis[N][2];
void add(int a, int b, int c) {
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++;
}
void djs() {
    for (int i = 1; i <= n; i++) dist[i][0] = INF, dist[i][1] = INF;
    for (int i = 1; i <= n; i++) vis[i][0] = 0, vis[i][1] = 0;
    for (int i = 1; i <= n; i++) cnt[i][0] = 0, cnt[i][1] = 0;
    dist[S][0] = 0, cnt[S][0] = 1;
    priority_queue<Ver, vector<Ver>, greater<Ver>> q;
    q.push({S, 0, 0});
    while (q.size()) {
        auto k = q.top();
        q.pop();
        int ver = k.id, type = k.type, count = cnt[ver][type], distance = k.dist;
        if (vis[ver][type]) continue;
        vis[ver][type] = 1;
        for (int i = h[ver]; i != -1; i = ne[i]) {
            int j = e[i];
            if (dist[j][0] > distance + w[i]) {
                dist[j][1] = dist[j][0], cnt[j][1] = cnt[j][0]; // 原最小值变次小值
                q.push({j, 1, dist[j][1]});
                dist[j][0] = distance + w[i];
                cnt[j][0] = count;
                q.push({j, 0, dist[j][0]});
            }
            else if (dist[j][0] == distance + w[i]) cnt[j][0] += count;
            else if (dist[j][1] > distance + w[i]) {
                dist[j][1] = distance + w[i];
                cnt[j][1] = count;
                q.push({j, 1, dist[j][1]});
            }
            else if (dist[j][1] == distance + w[i]) cnt[j][1] += count;
        }
    }
}
```

无向无权图

```
int n, m;
int h[N], e[M], ne[M], idx;
int d[N], cnt[N];
void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}
void bfs() {
    memset(d, 0x3f, sizeof(d));
    queue<int> q;
    d[1] = 0;
    cnt[1] = 1;
    q.push(1);
    while (q.size()) {
        auto k = q.front();
        q.pop();
        for (int i = h[k]; i != -1; i = ne[i]) {
            int j = e[i];
            if (d[j] > d[k] + 1) {
                d[j] = d[k] + 1;
                cnt[j] = cnt[k];
                q.push(j);
            }
            else if (d[j] == d[k] + 1) cnt[j] = (cnt[j] + cnt[k]) % mod;
        }
    }
}
void solve() {
    cin >> n >> m;
    memset(h, -1, sizeof(h));
    while (m--) {
        int a, b;
        cin >> a >> b;
        add(a, b), add(b, a);
    }
```

```
    bfs();
    for (int i = 1; i <= n; i++) cout << cnt[i] << '\n';
}
```

最小树形图

O(nm) 朱刘算法

不限定根结点的树形图，我们可以虚拟一个 0 号根节点，并向各个点连一条权值为 sum(w) + 1 的边，因为权值很大，所以最终结果一定只包含一条这样的边；但是如果答案大于 sum + sum + 1，其实是无解的，这样相当于两个点不联通，只好多选一条大边；否则答案是 ans − sum − 1

```
int n, m, rt, t, cnt, id[N], pre[N], ine[N], vis[N];
struct line {
    int x, y, c;
} q[M];
int zl() {
    int ans = 0;
    while (true) {
         cnt = 0;
         for (int i = 1; i <= n; ++i) ine[i] = INF, vis[i] = 0, id[i] = 0; // 预处理
         for (int i = 1; i <= m; ++i) if (q[i].x != q[i].y && ine[q[i].y] > q[i].c) ine[q[i].y] = q[i].c, pre[q[i].y] =
q[i].x; //每个点的最短边
         for (int i = 1; i <= n; ++i) if (i != rt && ine[i] == INF) return -1; // 有点无最短边
         for (int i = 1; i <= n; ++i) {
             if (i == rt) continue;
             ans += ine[i], t = i;
             while (vis[t] != i && !id[t] && t != rt) vis[t] = i, t = pre[t];
             // 能走到环的点或者换上的点停下
             if (!id[t] && t != rt) {
                id[t] = ++cnt; // 将环上的点标记为新的环
                for (int o = pre[t]; o != t; o = pre[o]) id[o] = cnt;
             }
         } // 找环
         if (!cnt) break; // 无环结束
         for (int i = 1; i <= n; ++i) if(!id[i]) id[i] = ++cnt;
         for (int i = 1; i <= m;++i) {
             t = q[i].y, q[i].x = id[q[i].x], q[i].y = id[q[i].y];
             if (q[i].x != q[i].y) q[i].c -= ine[t];
         }
         n = cnt, rt = id[rt];
         // 去旧图,换新图
    }
    return ans;
}
void solve() {
    cin >> n >> m >> rt;
    for (int i = 1; i <= m; i++) cin >> q[i].x >> q[i].y >> q[i].c;
    cout << zl() << '\n';
}
```

```
int n, m, r; // 点数，边数，根
LL d[N][N], bd[N][N];
int pre[N];
int dfn[N], low[N], ts, stk[N], top;
int id[N], cnt;
bool st[N], ins[N];
void dfs(int u) {
    st[u] = 1;
    for (int i = 1; i <= n; i++) {
        if (d[u][i] < INF && !st[i]) dfs(i);
    }
}
bool check_con() {
    dfs(r);
    for (int i = 1; i <= n; i++) {
        if (!st[i]) return 0;
    }
    return 1;
}
void tarjan(int u) {
    dfn[u] = low[u] = ++ts;
    stk[++top] = u, ins[u] = 1;
    int j = pre[u];
    if (!dfn[j]) {
        tarjan(j);
        low[u] = min(low[u], low[j]);
    }
    else if (ins[j]) low[u] = min(low[u], dfn[j]);
    if (low[u] == dfn[u]) {
```

```
        int y;
        ++cnt;
        do {
            y = stk[top--];
            ins[y] = 0;
            id[y] = cnt;
        } while (y != u);
    }
}
LL work() {
    LL res = 0;
    while (1) {
        for (int i = 1; i <= n; i++) {
            pre[i] = i;
            for (int j = 1; j <= n; j++) {
                if (d[pre[i]][i] > d[j][i]) {
                    pre[i] = j;
                }
            }
        }
        memset(dfn, 0, sizeof(dfn));
        ts = cnt = 0;
        for (int i = 1; i <= n; i++) {
            if (!dfn[i]) {
                tarjan(i);
            }
        }
        if (cnt == n) {
            for (int i = 1; i <= n; i++) {
                if (i != r) res += d[pre[i]][i];
            }
            break;
        }
        for (int i = 1; i <= n; i++) {
            if (i != r) {
                if (id[pre[i]] == id[i]) {
                    res += d[pre[i]][i];
                }
            }
        }
        for (int i = 1; i <= cnt; i++) {
            for (int j = 1; j <= cnt; j++) {
                bd[i][j] = INF;
            }
        }
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                if (d[i][j] < INF && id[i] != id[j]) {
                    int a = id[i], b = id[j];
                    if (id[pre[j]] == id[j]) bd[a][b] = min(bd[a][b], d[i][j] - d[pre[j]][j]);
                    else bd[a][b] = min(bd[a][b], d[i][j]);
                }
            }
        }
        n = cnt;
        r = id[r];
        memcpy(d, bd, sizeof(d));
    }
    return res;
}
void solve() {
    cin >> n >> m >> r;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            d[i][j] = INF;
        }
    }
    while (m--) {
        int a, b;
        LL c;
        cin >> a >> b >> c;
        if (a != b && b != r) d[a][b] = min(d[a][b], c);
    }
    if (!check_con()) cout << "-1" << '\n';
    else cout << work() << '\n';
}
```

bellman_ford

```
int n, m;       // n 表示点数，m 表示边数
```

```
int dist[N];        // dist[x] 存储 1 到 x 的最短路距离
int backup[N]       // 备份
bool flag;          // 用来确定是否存在最短路
struct Edge {  // 边，a 表示出点，b 表示入点，w 表示边的权重
    int a, b, w;
} edges[M];
// 求 1 到 n 的最短路距离，如果无法从 1 走到 n，。
int bellman_ford() {
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;
    // 如果第 n 次迭代仍然会松弛三角不等式，就说明存在一条长度是 n + 1 的最短路径，由抽屉原理，路径中至少存在两个相同的点，说明图中存在负权回路。
    for (int i = 0; i < n; i++) {
        memcpy(backup,dist,sizeof(dist));
        for (int j = 0; j < m; j++) {
            int a = edges[j].a, b = edges[j].b, w = edges[j].w;
            if (dist[b] > backup[a] + w) {
                dist[b] = backup[a] + w;
            }
        }
    }
    if (dist[n] > 0x3f3f3f3f / 2) flag = 1;
    return dist[n];
}
```

Boruvka

边具有较多特殊性质的问题中，Boruvka 算法具有优势

(1)计算每个点分别属于哪个连通块。将每个连通块都设为「没有最小边」。

(2)遍历每条边(u, v)，如果 u 和 v 不在同一个连通块，就用这条边的边权分别更新 u 和 v 所在连通块的最小边。

(3)如果所有连通块都没有最小边，退出程序，此时的 E' 就是原图最小生成森林的边集。否则，将每个有最小边的连通块的最小边加入 E'，返回第一步。

```
struct edge {
    int u, v, w;
} e[M];
int p[N], min_edge[N];
int find(int x) {
    if (p[x] != x) p[x] = find(p[x]);
    return p[x];
}
bool merge(int u, int v) {
    int fu = find(u), fv = find(v);
    if (fu == fv) return 0;
    p[fu] = fv;
    return 1;
}
void solve() {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) p[i] = i;
    for (int i = 1; i <= m; i++) cin >> e[i].u >> e[i].v >> e[i].w;
    int cnt_cmp = n;
    LL ans = 0;
    while (cnt_cmp > 1) {
        for (int i = 1; i <= n; i++) min_edge[i] = -1;
        for (int i = 1; i <= m; i++) {
            if (find(e[i].u) == find(e[i].v)) continue;
            int r_u = find(e[i].u);
            if (min_edge[r_u] == -1 || e[i].w < e[min_edge[r_u]].w) min_edge[r_u] = i;
            int r_v = find(e[i].v);
            if (min_edge[r_v] == -1 || e[i].w < e[min_edge[r_v]].w) min_edge[r_v] = i;
        }
        for (int i = 1; i <= n; i++) {
            if (min_edge[i] != -1) {
                if (merge(e[min_edge[i]].u, e[min_edge[i]].v)) {
                    ans += e[min_edge[i]].w;
                    cnt_cmp--;
                }
            }
        }
    }
    cout << ans << '\n';
}
```

djs

朴素 djs

```
int g[N][N];
int dist[N];
bool st[N];
int dijkstra() {
    memset(dist, 0x3f, sizeof(dist));
    dist[1] = 0;
    for (int i = 0; i < n - 1; i++) {
        int t = -1;     // 在还未确定最短路的点中，寻找距离最小的点
        for (int j = 1; j <= n; j++) {
            if (!st[j] && (t == -1 || dist[t] > dist[j])) {
                t = j;
            }
        }
        st[t] = true;
        // 用t更新其他点的距离
        for (int j = 1; j <= n; j++) dist[j] = min(dist[j], dist[t] + g[t][j]);
    }
    return dist[n];
}
```

堆优化的 djs

```
int h[N], e[M], ne[M], w[M], idx;
int d[N];
bool vis[N];
void add(int a, int b, int u) {
    e[idx] = b, w[idx] = u, ne[idx] = h[a], h[a] = idx++;
}
void djs(int u) {
    priority_queue<PII, vector<PII>, greater<PII>> q;
    d[u] = 0;
    q.push({d[u], u});
    while (q.size()) {
        auto k = q.top();
        q.pop();
        if (vis[k.second]) continue;
        vis[k.second] = 1;
        for (int i = h[k.second]; i != -1; i = ne[i]) {
            int j = e[i];
            if (d[j] > k.first + w[i]) {
                d[j] = k.first + w[i];
                q.push({d[j], j});
            }
        }
    }
}
```

floyd

多源最短路

```
int n, m;
int d[N][N];
void solve() {
    cin >> n >> m;
    memset(d, 0x3f, sizeof(d));
    for (int i = 1; i <= m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        d[a][b] = min(d[a][b], c), d[b][a] = min(d[b][a], c);
    }
    for (int i = 1; i <= n; i++) d[i][i] = 0;
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
            }
        }
    }
}
```

传递闭包

```
void floyd() {
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                d[i][j] |= d[i][k] && d[k][j];
            }
        }
    }
}
```

无向图最小环

```
int n, m;
int d[N][N], g[N][N];
int pos[N][N];
int path[N], cnt; // 最小环路径 最小环点数
int ans = INF; // 最小环之和
void get_path(int i, int j) {
    if (pos[i][j] == 0) return;
    int k = pos[i][j];
    get_path(i, k);
    path[cnt++] = k;
    get_path(k, j);
}
void solve() {
    cin >> n >> m;
    memset(g, 0x3f, sizeof(g));
    for (int i = 1; i <= n; i++) g[i][i] = 0;
    while (m--) {
        int a, b, c;
        cin >> a >> b >> c;
        g[a][b] = g[b][a] = min(g[a][b], c);
    }
    memcpy(d, g, sizeof(g));
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i < k; i++) {
            for (int j = i + 1; j < k; j++) {
                if ((LL)d[i][j] + g[j][k] + g[k][i] < ans) {
                    ans = (LL)d[i][j] + g[j][k] + g[k][i];
                    cnt = 0;
                    path[cnt++] = k;
                    path[cnt++] = i;
                    get_path(i, j);
                    path[cnt++] = j;
                }
            }
        }
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                if (d[i][j] > d[i][k] + d[k][j]) {
                    d[i][j] = d[i][k] + d[k][j];
                    pos[i][j] = k; // 转移
                }
            }
        }
    }
    if (ans == INF) cout << "No solution." << '\n';
    else {
        cout << ans << '\n';
        for (int i = 0; i < cnt; i++) cout << path[i] << ' ';
    }
}
```

从起点到终点恰好经过 k 条边的最短路（可重复经过边）

```
int n, m, k, S, E;
int g[N][N];
int res[N][N];
void mul(int c[][N], int a[][N], int b[][N]) {
    static int tmp[N][N];
    memset(tmp, 0x3f, sizeof(tmp));
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                tmp[i][j] = min(tmp[i][j], a[i][k] + b[k][j]);
            }
        }
    }
    memcpy(c, tmp, sizeof(tmp));
```

```
}
void qmi() {
    memset(res, 0x3f, sizeof(res));
    for (int i = 1; i <= n; i++) res[i][i] = 0;
    while (k) {
        if (k & 1) mul(res, res, g);
        mul(g, g, g);
        k >>= 1;
    }
}
void solve() {
    cin >> k >> m >> S >> E;
    memset(g, 0x3f, sizeof(g));
    map<int, int> ids;
    if (!ids.count(S)) ids[S] = ++n;
    if (!ids.count(E)) ids[E] = ++n;
    S = ids[S], E = ids[E];
    while (m--) {
        int a, b, c;
        cin >> c >> a >> b;
        if (!ids.count(a)) ids[a] = ++n;
        if (!ids.count(b)) ids[b] = ++n;
        a = ids[a], b = ids[b];
        g[a][b] = g[b][a] = min(g[a][b], c);
    }
    qmi();
    cout << res[S][E] << '\n';
}
```

Johnson

跑全源最短路，带负权，首先 spfa 判断负环然后建立一个虚拟节点，给每个点加一条 0 权值的边 spfa 求每个点的势能 djs 求最短路 真实最短路为的 d[i][j] - (h[i] - h[j]) 后者为势能

```
int n, m;
int h[N], e[M], ne[M], w[M], idx;
int d[N][N];
bool vis[N][N];
int cnt[N];
LL sum[N];
bool st[N];
int dd[N];
void add(int a,int b,int u) {
    e[idx] = b, w[idx] = u, ne[idx] = h[a], h[a] = idx++;
}
void spfa() {
    memset(dd, 0x3f, sizeof(dd));
    memset(st, 0, sizeof(st));
    queue<int> q;
    dd[0] = 0;
    st[0] = 1;
    q.push(0);
    while (q.size()) {
        auto k = q.front();
        q.pop();
        st[k] = 0;
        for (int i = h[k]; i != -1; i = ne[i]) {
            int j = e[i];
            if (dd[j] > dd[k] + w[i]) {
                dd[j] = dd[k] + w[i];
                if (!st[j]) {
                    q.push(j);
                    st[j] = 1;
                }
            }
        }
    }
}
bool spfa__fuhuan() {
    queue<int> q;
    for (int i = 1; i <= n ;i++) {
        q.push(i);
        st[i] = 1;
    }
    while (q.size()) {
        auto k = q.front();
        q.pop();
        st[k] = 0;
        for (int i = h[k]; i != -1; i = ne[i]) {
            int j = e[i];
```

```
            if (dd[j] > dd[k] + w[i]) {
                dd[j] = dd[k] + w[i];
                cnt[j] = cnt[k] + 1;
                if (cnt[j] >= n) return 1;
                if (!st[j]) {
                    q.push(j);
                    st[j] = 1;
                }
            }
        }
    }
    return 0;
}
void djs(int u) {
    priority_queue<PII,vector<PII>,greater<PII>> hh;
    d[u][u] = 0;
    hh.push({d[u][u],u});
    while (hh.size()) {
        auto k = hh.top();
        hh.pop();
        if (vis[u][k.second]) {
            continue;
        }
        vis[u][k.second] = 1;
        for (int i = h[k.second]; i != -1; i = ne[i]) {
            int j = e[i];
            if (d[u][j] > k.first + w[i] + dd[k.second] - dd[j]) {
                d[u][j] = k.first + w[i] + dd[k.second] - dd[j];
                hh.push({d[u][j],j});
            }
        }
    }
}
void solve() {
    cin >> n >> m;
    memset(h, -1, sizeof(h));
    for (int i = 1;i <= m; i++) {
        int x, y, u;
        cin >> x >> y >> u;
        add(x, y, u);
    }
    if (spfa__fuhuan()) {
        cout << -1;
        return;
    }
    memset(d, 0x3f, sizeof(d));
    for (int i = 1; i <= n; i++) add(0, i, 0);
    spfa();
    for (int i = 1; i <= n; i++) djs(i);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (d[i][j] == INF) sum[i] += (LL)j * 1e9;
            else sum[i] += (LL)j * (d[i][j] - (dd[i] - dd[j]));
        }
    }
    for (int i = 1; i <= n; i++) cout << sum[i] << '\n';
}
```

K短路

A*算法, 不是最优解

```
typedef pair<int, PII> PIII;
int n, m, S, T, K;  // 点，边，起点，终点
int h[N], rh[N], e[M], ne[M], w[M], idx;
int dist[N], cnt[N]; // cnt计数每个点第几次遇到
bool vis[N];
void add(int h[], int a, int b, int c) {
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++;
}
void djs() {
    priority_queue<PII, vector<PII>, greater<PII>> heap;
    heap.push({0, T});
    memset(dist, 0x3f, sizeof(dist));
    dist[T] = 0;
    while (heap.size()) {
        auto k = heap.top();
        heap.pop();
        if (vis[k.second]) continue;
        vis[k.second] = 1;
```

```
        for (int i = rh[k.second]; i != -1; i = ne[i]) {
            int j = e[i];
            if (dist[j] > k.first + w[i]) {
                dist[j] = k.first + w[i];
                heap.push({dist[j], j});
            }
        }
    }
}
int a_star() {
    priority_queue<PIII, vector<PIII>, greater<PIII>> heap;
    heap.push({dist[S], {0, S}});
    while (heap.size()) {
        auto k = heap.top();
        heap.pop();
        int ver = k.second.second, distance = k.second.first;
        cnt[ver]++;
        if (cnt[T] == K) return distance;
        for (int i = h[ver]; i != -1; i = ne[i]) {
            int j = e[i];
            if (cnt[j] < K) heap.push({distance + w[i] + dist[j], {distance + w[i], j}});
        }
    }
    return -1;
}
void solve() {
    cin >> n >> m;
    memset(h, -1, sizeof(h));
    memset(rh, -1, sizeof(rh));
    for (int i = 0; i < m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        add(h, a, b, c);
        add(rh, b, a, c);
    }
    cin >> S >> T >> K;
    if (S == T) K++; // 起点和终点重合，且必须有边
    djs();
    cout << a_star() << '\n';
}
```

Kruskal

最小生成树

O(mlog(m)) 适合稀疏图

```
int n, m;
int p[N];
int ra[N];
struct Edge {
    int a, b, w;
    bool operator< (const Edge &W) const {
        return w < W.w;
    }
} e[M];
int find(int x) {
    if (p[x] != x) p[x] = find(p[x]);
    return p[x];
}
int kruskal() {
    sort(e, e + m);
    for (int i = 1; i <= n; i++) p[i] = i;
    int res = 0, cnt = 0;
    for (int i = 0; i < m; i++) {
        int a = edges[i].a, b = edges[i].b, w = edges[i].w;
        a = find(a), b = find(b);
        if (a != b) {
            if (ra[a] <= ra[b]) {
                p[a] = b;
                if (ra[a] == ra[b]) ra[b]++;
            }
            else p[b] = a;
            res += w;
            cnt++ ;
        }
    }
    if (cnt < n - 1) return INF;
    return res;
}
```

Kruskal 重构树

原图中两个点之间的所有简单路径上最大边权的最小值 = 最小生成树上两个点之间的简单路径上的最大值 = Kruskal 重构树上两点之间的 LCA 的权值。

原图中两个点之间的所有简单路径上最小边权的最大值 则从大到小加边

原图中两个点之间的所有简单路径上最大边权的最小值 则从小到大加边

```
int n, m, q;
int p[N], a[N], f[N][20], dep[N];
bool vis[N];
vector<int> E[N];
struct edge {
    int u, v, w;
} e[N];
bool cmp(edge a, edge b) {
  return a.w > b.w;
}
int find(int x) {
    if (p[x] != x) p[x] = find(p[x]);
    return p[x];
}
void dfs(int u, int fa) {
  vis[u] = 1;
  f[u][0] = fa;
  dep[u] = dep[fa] + 1;
  for(auto &v : E[u]) {
    if(v == fa) continue;
    dfs(v, u);
  }
}
int lca(int u, int v) {
  if (dep[u] < dep[v]) swap(u, v);
  int d = dep[u] - dep[v];
  for (int i = 19; i >= 0;i--) {
    if (d >> i & 1) {
            u = f[u][i];
        }
  }
  if(u == v) return u;
  for (int i = 19; i >= 0;i--) {
    if (f[u][i] != f[v][i]) {
            u = f[u][i], v = f[v][i];
        }
  }
  return f[u][0];
}
void solve() {
    cin >> n >> m >> q;
    for (int i = 1; i <= n + m; i++) p[i] = i;
    for (int i = 1; i <= m; i++) cin >> e[i].u >> e[i].v >> e[i].w;
    sort(e + 1, e + m + 1, cmp);
    int cur = n;
  for(int i = 1; i <= m; i++) {
    int u = e[i].u, v = e[i].v, w = e[i].w;
    u = find(u), v = find(v);
    if (u == v) continue;
    ++cur;
    p[u] = p[v] = cur;
    E[cur].push_back(u);
    E[cur].push_back(v);
    a[cur] = w;
  }
  for (int i = cur; i >= 1; i--) {
    if (!vis[i]) {
            dfs(i, 0);
        }
  }
  for (int j = 1; j < 20; j++) {
    for (int i = 1; i <= cur; i++) {
      f[i][j] = f[f[i][j - 1]][j - 1];
    }
  }
    while (q--) {
        int u, v;
    cin >> u >> v;
    if (find(u) != find(v)) {
      cout << -1 << '\n';
      continue;
```

```
    }
    cout << a[lca(u, v)] << '\n';
  }
}
```

LCA

倍增求lca

```
int n, m, root;
int depth[N], fa[N][20];
vector<int> adj[N];
void bfs(int root) {
    memset(depth, 0x3f, sizeof(depth));
    depth[0] = 0, depth[root] = 1;
    queue<int> q;
    q.push(root);
    while (q.size()) {
        auto k = q.front();
        q.pop();
        for (auto v : adj[k]) {
            if (depth[v] > depth[k] + 1) {
                depth[v] = depth[k] + 1;
                q.push(v);
                fa[v][0] = k;
                for (int t = 1; t <= 19; t++) fa[v][t] = fa[fa[v][t - 1]][t - 1];
            }
        }
    }
}
int lca(int a, int b) {
    if (depth[a] < depth[b]) swap(a, b);
    for (int k = 19; k >= 0; k--) {
        if (depth[fa[a][k]] >= depth[b]) a = fa[a][k];
    }
    if (a == b) return a;
    for (int k = 19; k >= 0; k--) {
        if (fa[a][k] != fa[b][k]) {
            a = fa[a][k];
            b = fa[b][k];
        }
    }
    return fa[a][0];
}
```

Prim

朴素 Prim O(n^2) 适合稠密图

```
int n;
int g[N][N];    // 邻接矩阵，存储所有边
int dist[N];    // 存储其他点到当前最小生成树的距离
bool vis[N];     // 存储每个点是否已经在生成树中
// 如果图不连通，则返回 INF，否则返回最小生成树的树边权重之和
int prim() {
    memset(dist, 0x3f, sizeof(dist));
    dist[1] = 0;
    int res = 0;
    for (int i = 1; i <= n; i++) {
        int t = -1;
        for (int j = 1; j <= n; j++) {
            if (!vis[j] && (t == -1 || dist[t] > dist[j])) {
                t = j;
            }
        }
        if (i && dist[t] == INF) return INF;
        if (i) res += dist[t];
        vis[t] = true;
        for (int j = 1; j <= n; j ++ ) dist[j] = min(dist[j], g[t][j]);
    }
    return res;
}
```

堆优化 prim 不如 kruskal

```
int n, m;
int h[N], e[M], ne[M], w[M], idx;
int d[N];
```

```
bool vis[N];
LL sum = 0;
int cnt = 0;
void add(int a, int b, int u) {
    e[idx] = b, w[idx] = u, ne[idx] = h[a], h[a] = idx++;
}
void prim() {
    memset(d, 0x3f, sizeof(d));
    priority_queue<PII, vector<PII>, greater<PII>> hh;
    d[1] = 0;
    hh.push({0, 1});
    while (hh.size()) {
        auto k = hh.top();
        hh.pop();
        if (vis[k.second]) continue;
        vis[k.second] = 1;
        if (k.first != INF) {
            cnt++;
            sum += k.first;
        }
        for (int i = h[k.second]; i != -1; i = ne[i]) {
            int j = e[i];
            if (d[j] > w[i]) {
                d[j] = w[i];
                hh.push({d[j],j});
            }
        }
    }
}
void solve() {
    cin >> n >> m;
    memset(h, -1, sizeof(h));
    for (int i = 1; i <= m; i++) {
        int x, y, u;
        cin >> x >> y >> u;
        add(x, y, u), add(y, x, u);
    }
    prim();
    if (cnt != n) cout << -1 << '\n';
    else cout << sum << '\n';
}
```

prufer

将一个带标号 n 个结点的树用 [1, n] 中的 n - 2 个整数表示

完全图有 n^(n - 2) 棵生成树

k 个连通块添加 k - 1 条边连通的方案数为 n^(k-2) * s1 * s2 * s3... * sk

```
int n, m;
int f[N], d[N], p[N];
void tree2prufer() {
    for (int i = 1; i < n; i++) {
        cin >> f[i];
        d[f[i]]++;
    }
    for (int i = 0, j = 1; i < n - 2; j++) {
        while (d[j]) j++;
        p[i++] = f[j];
        while (i < n - 2 && --d[p[i - 1]] == 0 && p[i - 1] < j) p[i++] = f[p[i - 1]];
    }
    for (int i = 0; i < n - 2; i++) cout << p[i] << ' ';
}
void prufer2tree() {
    for (int i = 1; i <= n - 2; i++) {
        cin >> p[i];
        d[p[i]]++;
    }
    p[n - 1] = n;
    for (int i = 1, j = 1; i < n; i++, j++) {
        while (d[j]) j++;
        f[j] = p[i];
        while (i < n - 1 && --d[p[i]] == 0 && p[i] < j) f[p[i]] = p[i + 1], i++;
    }
    for (int i = 1; i <= n - 1; i++) cout << f[i] << ' ';
}
void solve() {
    cin >> n >> m;
    if (m == 1) tree2prufer();
```

```
    else prufer2tree();
}
```

SPFA

SPFA 求最短路

最坏O(n * m)

```
int n, m, st, ed; // 点，边，起点，终点
int h[N], e[M], ne[M], w[M], idx;
int dist[N];
bool vis[N];
void add(int a, int b, int c) {
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++;
}
void spfa(int u) { // u为起点
    queue<int> q;
    memset(dist, 0x3f, sizeof(dist));
    dist[u] = 0;
    vis[u] = 1;
    q.push(u);
    while (!q.empty()) {
        auto k = q.front();
        q.pop();
        vis[k] = 0;
        for (int i = h[k]; i != -1; i = ne[i]) {
            int j = e[i];
            if (dist[j] > dist[k] + w[i]) {
                dist[j] = dist[k] + w[i];
                if (!vis[j]) {
                    q.push(j);
                    vis[j] = 1;
                }
            }
        }
    }
}
void solve() {
    cin >> n >> m >> st >> ed;
    memset(h, -1, sizeof(h));
    for (int i = 1; i <= m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c), add(b, a, c);
    }
    spfa(st);
    cout << dist[ed] << '\n';
}
```

判断负环

```
bool spfa() {
    memset(dist, 0x3f, sizeof(dist));
    memset(cnt, 0, sizeof(cnt));
    stack<int> q;
    for (int i = 1; i <= n ;i++) {
        q.push(i);
        vis[i] = 1;
    }
    while (q.size()) {
        auto k = q.top();
        q.pop();
        vis[k] = 0;
        for (int i = h[k]; i != -1; i = ne[i]) {
            int j = e[i];
            if (dist[j] > dist[k] + w[i]) { // 找正环时将不等号改为小于即可
                dist[j] = dist[k] + w[i];
                cnt[j] = cnt[k] + 1;
                if (cnt[j] >= n) return 1;
                if (!vis[j]) {
                    q.push(j);
                    vis[j] = 1;
                }
            }
        }
    }
    return 0;
}
```

数据结构

并查集

```
struct DSU {
    vector<int> fa, dis;
    DSU(int n) {
        fa.assign(n, 0);
        dis.assign(n, 0);
        iota(fa.begin(), fa.end(), 0);
    }
    int find(int u) {
        if (fa[u] == u) return u;
        int f = find(fa[u]);
        dis[u] += dis[fa[u]];
        return fa[u] = f;
    }
    void merge(int u, int v) {
        int f = find(u);
        dis[v] = dis[u] + 1;
        fa[v] = f;
    }
    int dist(int u) {
        find(u);
        return dis[u];
    }
};
```

单调栈

可求左边距离最近的比当前值大的一个元素

```
int stk[N], tt;
for (int i = 1; i <= n; i++) {
    cin >> x;
    while (tt && stk[tt] >= x) tt--;
    if (tt) {
        cout << stk[tt] << ' ';
    }
    else cout << -1 << ' ';
    stk[++tt] = x;
}
```

单调队列

找出滑动窗口中的最大值/最小值

```
int hh = 0, tt = -1;
for (int i = 0; i < n; i ++ ) {
    while (hh <= tt && check_out(q[hh])) hh++;  // 判断队头是否滑出窗口
    while (hh <= tt && check(q[tt], i)) tt--;
    q[++tt] = i;
}
```

求 k * k 的矩阵中的最大值和最小值

二维单调队列

```
int d[N][N];
int row_max[N][N], row_min[N][N];
int a[N], b[N], c[N];
int q[N];
int n, m, k;
void get_min(int a[], int b[], int tot) {
    int hh = 0, tt = -1;
    for (int i = 1; i <= tot; i++) {
        if (hh <= tt && q[hh] <= i - k) hh++;
        while (hh <= tt && a[q[tt]] >= a[i]) tt--;
        q[++tt] = i;
        b[i] = a[q[hh]];
    }
}
void get_max(int a[], int b[], int tot) {
    int hh = 0, tt = -1;
    for (int i = 1; i <= tot; i++) {
        if (hh <= tt && q[hh] <= i - k) hh++;
        while (hh <= tt && a[q[tt]] <= a[i]) tt--;
```

```
        q[++tt] = i;
        b[i] = a[q[hh]];
    }
}
void solve() {
    cin >> n >> m >> k;
    int res = INT_MAX;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> d[i][j];
        }
    }
    for (int i = 1; i <= n; i++) {
        get_min(d[i], row_min[i], m);
        get_max(d[i], row_max[i], m);
    }
    for (int i = k; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            a[j] = row_min[j][i];
        }
        get_min(a, b, n);
        for (int j = 1; j <= n; j++) {
            a[j] = row_max[j][i];
        }
        get_max(a, c, n);
        for (int j = k; j <= n; j++) {
            res = min(res, c[j] - b[j]);
        }
    }
    cout << res << '\n';
}
```

笛卡尔树

```
struct Node {
    int idx, val;       // 节点索引 节点的值
    int par, ch[2];     // 父节点索引 左右子节点索引
    void init(int _idx, int _val, int _par) {
        idx = _idx, val = _val, par = _par, ch[0] = ch[1] = 0;
    }
} tree[N];
int root;
int cartesian_build(int n) {
    for (int i = 1; i <= n; i++) {
        int k = i - 1;
        // 小根堆
        while (tree[k].val > tree[i].val) k = tree[k].par;
        // 大根堆
        //while (k > 0 && tree[k].val < tree[i].val) k = tree[k].par;
        tree[i].ch[0] = tree[k].ch[1];
        tree[k].ch[1] = i;
        tree[i].par = k;
        tree[tree[i].ch[0]].par = i;
    }
    return tree[0].ch[1];
}
void solve() {
    int n;
    cin >> n;
    tree[0].init(0, 0, 0);
    for (int i = 1; i <= n; i++) {
        int x;
        cin >> x;
        tree[i].init(i, x, 0);
    }
    root = cartesian_build(n);
}
```

珂朵莉树

```
#define IT set<node>::iterator
struct node {
    int l, r;
    mutable int v;
    bool operator < (const node &x) const {
        return l < x.l;
    }
    node (int L, int R, int V) {
        l = L, r = R, v = V;
```

```
    }
};
set<node> s;
IT split(int pos) {
    IT it = s.lower_bound(node(pos, 0, 0));
    if (it != s.end() && it->l == pos) return it;
    it--;
    int L = it->l, R = it->r, V = it->v;
    s.erase(it);
    s.insert(node(L, pos - 1, V));
    return s.insert(node(pos, R, V)).first;
}
void tp(int l, int r, int v) {
    IT itr = split(r + 1), itl = split(l);
    s.erase(itl, itr);
    s.insert(node(l, r, v));
}
int ask(int l, int r) {
    set<int> cnt;
    IT itr = split(r + 1), itl = split(l);
    for (IT it = itl; it != itr; it++) {
        cnt.insert(it->v);
    }
    return cnt.size();
}
void solve() {
    int q;
    cin >> q;
    s.insert(node(1, 100002, 0));
    while (q--) {
        int l, r, x;
        cin >> l >> r >> x;
        tp(l, r, x);
        cout << ask(1, 100002) << '\n';
    }
}
```

科技

```
struct BitRank {
    // block 管理一行一行的 bit
    std::vector<unsigned long long> block;
    std::vector<unsigned int> count;
    BitRank() {}
    // 位向量长度
    void resize(const unsigned int num) {
        block.resize(((num + 1) >> 6) + 1, 0);
        count.resize(block.size(), 0);
    }
    // 设置 i 位bit
    void set(const unsigned int i, const unsigned long long val) {
        block[i >> 6] |= (val << (i & 63));
    }
    void build() {
        for (unsigned int i = 1; i < block.size(); i++) {
            count[i] = count[i - 1] + __builtin_popcountll(block[i - 1]);
        }
    }
    // [0, i) 1 的个数
    unsigned int rank1(const unsigned int i) const {
        return count[i >> 6] +
            __builtin_popcountll(block[i >> 6] & ((1ULL << (i & 63)) - 1ULL));
    }
    // [i, j) 1 的个数
    unsigned int rank1(const unsigned int i, const unsigned int j) const {
        return rank1(j) - rank1(i);
    }
    // [0, i) 0 的个数
    unsigned int rank0(const unsigned int i) const { return i - rank1(i); }
    // [i, j) 0 的个数
    unsigned int rank0(const unsigned int i, const unsigned int j) const {
        return rank0(j) - rank0(i);
    }
};

class WaveletMatrix {
private:
    unsigned int height;
    std::vector<BitRank> B;
    std::vector<int> pos;
```

```
public:
    WaveletMatrix() {}
    WaveletMatrix(std::vector<int> vec)
        : WaveletMatrix(vec, *std::max_element(vec.begin(), vec.end()) + 1) {}
    // sigma: 字母表大小(字符串的话)，数字序列的话是数的种类
    WaveletMatrix(std::vector<int> vec, const unsigned int sigma) {
        init(vec, sigma);
    }
    void init(std::vector<int>& vec, const unsigned int sigma) {
        height = (sigma == 1) ? 1 : (64 - __builtin_clzll(sigma - 1));
        B.resize(height), pos.resize(height);
        for (unsigned int i = 0; i < height; ++i) {
            B[i].resize(vec.size());
            for (unsigned int j = 0; j < vec.size(); ++j) {
                B[i].set(j, get(vec[j], height - i - 1));
            }
            B[i].build();
            auto it = stable_partition(vec.begin(), vec.end(), [&](int c) {
                return !get(c, height - i - 1);
            });
            pos[i] = it - vec.begin();
        }
    }

    int get(const int val, const int i) { return val >> i & 1; }
    // [l, r) 中 val 出现的频率

    int rank(const int val, const int l, const int r) {
        return rank(val, r) - rank(val, l);
    }
    // [0, i) 中 val 出现的频率
    int rank(int val, int i) {
        int p = 0;
        for (unsigned int j = 0; j < height; ++j) {
            if (get(val, height - j - 1)) {
                p = pos[j] + B[j].rank1(p);
                i = pos[j] + B[j].rank1(i);
            } else {
                p = B[j].rank0(p);
                i = B[j].rank0(i);
            }
        }
        return i - p;
    }
    // [l, r) 中第 k 小
    int quantile(int k, int l, int r) {
        int res = 0;
        for (unsigned int i = 0; i < height; ++i) {
            const int j = B[i].rank0(l, r);
            if (j > k) {
                l = B[i].rank0(l);
                r = B[i].rank0(r);
            } else {
                l = pos[i] + B[i].rank1(l);
                r = pos[i] + B[i].rank1(r);
                k -= j;
                res |= (1 << (height - i - 1));
            }
        }
        return res;
    }
    int rangefreq(const int i, const int j, const int a, const int b, const int l,
                  const int r, const int x) {
        if (i == j || r <= a || b <= l) return 0;
        const int mid = (l + r) >> 1;
        if (a <= l && r <= b) {
            return j - i;
        } else {
            const int left =
                rangefreq(B[x].rank0(i), B[x].rank0(j), a, b, l, mid, x + 1);
            const int right = rangefreq(pos[x] + B[x].rank1(i),
                                        pos[x] + B[x].rank1(j), a, b, mid, r, x + 1);
            return left + right;
        }
    }
    // [l,r) 在 [a, b) 值域的数字个数
    int rangefreq(const int l, const int r, const int a, const int b) {
        return rangefreq(l, r, a, b, 0, 1 << height, 0);
    }
    int rangemin(const int i, const int j, const int a, const int b, const int l,
```

```
                const int r, const int x, const int val) {
        if (i == j || r <= a || b <= l) return -1;
        if (r - l == 1) return val;
        const int mid = (l + r) >> 1;
        const int res =
            rangemin(B[x].rank0(i), B[x].rank0(j), a, b, l, mid, x + 1, val);
        if (res < 0)
            return rangemin(pos[x] + B[x].rank1(i), pos[x] + B[x].rank1(j), a, b, mid,
                            r, x + 1, val + (1 << (height - x - 1)));
        else
            return res;
    }
    // [l,r) 在 [a,b) 值域内存在的最小值是什么，不存在返回 -1
    int rangemin(int l, int r, int a, int b) {
        return rangemin(l, r, a, b, 0, 1 << height, 0, 0);
    }
};

int n, q;
void solve() {
    cin >> n >> q;
    vector<int> a(n + 1);
    for (int i = 1; i <= n; i++) cin >> a[i];
    WaveletMatrix wm(a);
    while (q--) {
        int l, r;
        cin >> l >> r;
        int k = (r - l + 1) / 2;
        r++;
        cout << wm.quantile(k, l, r) << '\n'; // 第 k 小以 0 为下标
    }
}
```

可持久化数据结构

可持久化Trie

每次只修改被添加或值被修改的节点，而保留没有被改动的节点，在上一个版本的基础上连边，使最后每个版本的 Trie 树的根遍历所能分离出的 Trie 树都是完整且包含全部信息的。

1:序列末尾插入一个数 x

2:询问 [l, r] 中的一个 p 位置使得后缀异或 x 最大

```
int n, m;
int s[N];
int tr[M][2], max_id[M];
int root[N], idx;
void insert(int i, int k, int p, int q) {
    if (k < 0) {
        max_id[q] = i;
        return;
    }
    int v = s[i] >> k & 1;
    if (p) tr[q][v ^ 1] = tr[p][v ^ 1];
    tr[q][v] = ++idx;
    insert(i, k - 1, tr[p][v], tr[q][v]);
    max_id[q] = max(max_id[tr[q][0]], max_id[tr[q][1]]);
}
int query(int root, int C, int L) {
    int p = root;
    for (int i = 23; i >= 0; i--) {
        int v = C >> i & 1;
        if (max_id[tr[p][v ^ 1]] >= L) p = tr[p][v ^ 1];
        else p = tr[p][v];
    }
    int ans = C ^ s[max_id[p]];
    return ans;
}
void solve() {
    cin >> n >> m;
    max_id[0] = -1;
    root[0] = ++idx;
    insert(0, 23, 0, root[0]);
    for (int i = 1; i <= n; i++) {
        int x;
        cin >> x;
        s[i] = s[i - 1] ^ x;
        root[i] = ++idx;
        insert(i, 23, root[i - 1], root[i]);
```

```
    }
    string op;
    int l, r, x;
    while (m--) {
        cin >> op;
        if (op[0] == 'A') {
            cin >> x;
            n++;
            s[n] = s[n - 1] ^ x;
            root[n] = ++idx;
            insert(n, 23, root[n - 1], root[n]);
        }
        else {
            cin >> l >> r >> x;
            cout << query(root[r - 1], s[n] ^ x, l - 1) << '\n';
        }
    }
}
```

主席树

查询区间第 k 小值

```
int n, m;
int a[N];
vector<int> nums;
struct Node {
    int l, r;
    int cnt; // 区间元素个数
} tr[N * 4 + N * 17]; // N * 4 + NlogN
int root[N], idx;
int find(int x) {
    return lower_bound(nums.begin(), nums.end(), x) - nums.begin();
}
int build(int l, int r) {
    int p = ++idx;
    if (l == r) return p;
    int mid = l + r >> 1;
    tr[p].l = build(l, mid), tr[p].r = build(mid + 1, r);
    return p;
}
int insert(int p, int l, int r, int x) {
    int q = ++idx;
    tr[q] = tr[p];
    if (l == r) {
        tr[q].cnt++;
        return q;
    }
    int mid = l + r >> 1;
    if (x <= mid) tr[q].l = insert(tr[p].l, l, mid, x);
    else tr[q].r = insert(tr[p].r, mid + 1, r, x);
    tr[q].cnt = tr[tr[q].l].cnt + tr[tr[q].r].cnt;
    return q;
}
int query(int q, int p, int l, int r, int k) {
    if (l == r) return r;
    int cnt = tr[tr[q].l].cnt - tr[tr[p].l].cnt;
    int mid = l + r >> 1;
    if (k <= cnt) return query(tr[q].l, tr[p].l, l, mid, k);
    else return query(tr[q].r, tr[p].r, mid + 1, r, k - cnt);
}
void solve() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        nums.push_back(a[i]);
    }
    sort(nums.begin(), nums.end());
    nums.erase(unique(nums.begin(), nums.end()), nums.end());
    root[0] = build(0, nums.size() - 1);
    for (int i = 1; i <= n; i++) root[i] = insert(root[i - 1], 0, nums.size() - 1, find(a[i]));
    while (m--) {
        int l, r, k;
        cin >> l >> r >> k;
        cout << nums[query(root[r], root[l - 1], 0, nums.size() - 1, k)] << '\n';
    }
}
```

李超线段树

```
struct LiChao {
    struct Line {
        long long k, b;                    // y = k·x + b
        Line(long long _k = 0, long long _b = (1LL << 62)) : k(_k), b(_b) {}
        long long operator()(long long x) const { return k * x + b; }
    };
    struct Node {
        Line line;
        int  ch[2]{0, 0};                   // 左 右
    };
    vector<Node> tr;
    int root = 0;
    const int X_MIN, X_MAX;                // 定义坐标范围
    LiChao(int l, int r) : X_MIN(l), X_MAX(r) {
        tr.reserve(10'000'000);
        tr.push_back(Node{});
    }
    int new_node(Line ln) {
        tr.push_back({ln, {0, 0}});
        return (int)tr.size() - 1;
    }
    /* --- 单点插直线 --- */
    void add_line(Line ln) { root = add_line(root, X_MIN, X_MAX, ln); }
    int add_line(int id, int l, int r, Line ln) {
        if(!id) { id = new_node(ln); return id; }
        int mid = (l + r) >> 1;
        bool lefBetter = ln(l) < tr[id].line(l);
        bool midBetter = ln(mid) < tr[id].line(mid);
        if (midBetter) swap(tr[id].line, ln);
        if (l == r) return id;
        if (lefBetter != midBetter) tr[id].ch[0] = add_line(tr[id].ch[0], l, mid, ln);        // ★ 写回
        else tr[id].ch[1] = add_line(tr[id].ch[1], mid+1, r, ln);      // ★ 写回
        return id;
    }
    /* --- 区间插直线（可选）--- */
    void add_segment(Line ln, int L, int R) {
        root = add_segment(root, X_MIN, X_MAX, ln, L, R);
    }
    int add_segment(int id, int l, int r, Line ln, int L, int R) {
        if (R < l || r < L) return id;
        if (L <= l && r <= R) return add_line(id, l, r, ln);
        int mid = (l + r) >> 1;
        tr[id].ch[0] = add_segment(tr[id].ch[0], l, mid, ln, L, R);
        tr[id].ch[1] = add_segment(tr[id].ch[1], mid + 1, r, ln, L, R);
        return id;
    }
    /* --- 查询点（最小值）--- */
    long long query(long long x) { return query(root, X_MIN, X_MAX, x); }
    long long query(int id, int l, int r, long long x) const {
        if (!id) return (1LL << 62);                 // +INF
        if (l == r) return tr[id].line(x);
        int mid = (l + r) >> 1;
        long long cur = tr[id].line(x);
        if (x <= mid) return min(cur, query(tr[id].ch[0], l, mid, x));
        else return min(cur, query(tr[id].ch[1], mid + 1, r, x));
    }
    /* --- 清空整棵树（多组数据时建议重建对象） --- */
    void clear() {
        tr.clear(); tr.shrink_to_fit();
        root = 0;
    }
};
```

莫队

对于序列上的区间询问问题，如果从 [l, r] 的答案能够 O(1) 扩展到 [l - 1, r], [l + 1, r], [l, r + 1], [l, r - 1]（即与 [l, r] 相邻的区间）的答案，那么可以在 O(n sqrtn) 的复杂度内求出所有询问的答案。

普通莫队

S = n / sqrt(m)

区间种类数

```
int n, m, len;
int w[N], ans[M];
struct Query {
    int id, l, r;
} q[M];
int cnt[S];
```

```
int get(int x) {
    return x / len;
}
bool cmp(Query a, Query b) {
    int i = get(a.l), j = get(b.l);
    if (i != j) return i < j;
    else {
        if (i & 1) return a.r < b.r;
        else return a.r > b.r;
    }
}
void add(int x, int &res) {
    if (!cnt[x]) res++;
    cnt[x]++;
}
void del(int x, int &res) {
    cnt[x]--;
    if (!cnt[x]) res--;
}
void solve() {
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> w[i];
    cin >> m;
    len = max(1, int((double)n / sqrt(m)));
    for (int i = 0; i < m; i++) {
        int l, r;
        cin >> l >> r;
        q[i] = {i, l, r};
    }
    sort(q, q + m, cmp);
    int res = 0;
    for (int k = 0, i = 0, j = 1; k < m; k++) {
        int id = q[k].id, l = q[k].l, r = q[k].r;
        while (i < r) add(w[++i], res);
        while (i > r) del(w[i--], res);
        while (j < l) del(w[j++], res);
        while (j > l) add(w[--j], res);
        ans[id] = res;
    }
    for (int i = 0; i < m; i++) cout << ans[i] << '\n';
}
```

带修莫队

S = n ^ (2 / 3)

在基础莫队上添加一维时间

求区间内共有几个不同的数，可单点修改一个数

```
int n, m, len, nq, nm;
int w[N], cnt[S], ans[N];
struct Query {
    int id, l, r, t;
} q[N];
struct Modify {
    int p, c;
} c[N];
int get(int x) {
    return x / len;
}
bool cmp(Query a, Query b) {
    int al = get(a.l), ar = get(a.r);
    int bl = get(b.l), br = get(b.r);
    if (al != bl) return al < bl;
    if (ar != br) return ar < br;
    return a.t < b.t;
}
void add(int x, int& res) {
    if (!cnt[x]) res++;
    cnt[x]++;
}
void del(int x, int& res) {
    cnt[x]--;
    if (!cnt[x]) res--;
}
void solve() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> w[i];
    for (int i = 0; i < m; i++) {
```

```
        char op[2];
        int a, b;
        cin >> op >> a >> b;
        if (*op == 'Q')  {
            nq++;
            q[nq] = {nq, a, b, nm};
        }
        else c[++nm] = {a, b};
    }
    len = cbrt((double)n * max(1 , nm)) + 1;
    sort(q + 1, q + nq + 1, cmp);
    int res = 0;
    for (int i = 0, j = 1, t = 0, k = 1; k <= nq; k++) {
        int id = q[k].id, l = q[k].l, r = q[k].r, tm = q[k].t;
        while (i < r) add(w[++i], res);
        while (i > r) del(w[i--], res);
        while (j < l) del(w[j++], res);
        while (j > l) add(w[--j], res);
        while (t < tm) {
            t++;
            if (c[t].p >= j && c[t].p <= i) {
                del(w[c[t].p], res);
                add(c[t].c, res);
            }
            swap(w[c[t].p], c[t].c);
        }
        while (t > tm) {
            if (c[t].p >= j && c[t].p <= i) {
                del(w[c[t].p], res);
                add(c[t].c, res);
            }
            swap(w[c[t].p], c[t].c);
            t--;
        }
        ans[id] = res;
    }
    for (int i = 1; i <= nq; i++) cout << ans[i] << '\n';
}
```

回滚莫队

只有增加不可实现或者只有删除不可实现的时候，就可以使用回滚莫队在 O(n sqrt m) 的时间内解决问题。回滚莫队的核心思想就是：既然只能实现一个操作，那么就只使用一个操作，剩下的交给回滚解决。

查询区间个数乘以当前数的最大值

```
int n, m, len;
int w[N], cnt[N];
LL ans[N];
struct Query {
    int id, l, r;
} q[N];
vector<int> nums;
int get(int x) {
    return x / len;
}
bool cmp(Query a, Query b) {
    int i = get(a.l), j = get(b.l);
    if (i != j) return i < j;
    else return a.r < b.r;
}
void add(int x, LL& res) {
    cnt[x]++;
    res = max(res, (LL)cnt[x] * nums[x]);
}
void solve() {
    cin >> n >> m;
    len = sqrt(n) + 1;
    for (int i = 1; i <= n; i++) {
        cin >> w[i];
        nums.push_back(w[i]);
    }
    sort(nums.begin(), nums.end());
    nums.erase(unique(nums.begin(), nums.end()), nums.end());
    for (int i = 1; i <= n; i++) w[i] = lower_bound(nums.begin(), nums.end(), w[i]) - nums.begin();
    for (int i = 0; i < m; i++) {
        int l, r;
        cin >> l >> r;
        q[i] = {i, l, r};
    }
```

```
    sort(q, q + m, cmp);
    for (int x = 0; x < m;) {
        int y = x;
        while (y < m && get(q[y].l) == get(q[x].l)) y++;
        int right = get(q[x].l) * len + len - 1;
        // 暴力求块内的询问
        while (x < y && q[x].r <= right) {
            LL res = 0;
            int id = q[x].id, l = q[x].l, r = q[x].r;
            for (int k = l; k <= r; k++) add(w[k], res);
            ans[id] = res;
            for (int k = l; k <= r; k++) cnt[w[k]]--;
            x++;
        }
        // 求块外的询问
        LL res = 0;
        int i = right, j = right + 1;
        while (x < y) {
            int id = q[x].id, l = q[x].l, r = q[x].r;
            while (i < r) add(w[++i], res);
            LL backup = res;
            while (j > l) add(w[--j], res);
            ans[id] = res;
            while (j < right + 1) cnt[w[j++]]--;
            res = backup;
            x++;
        }
        memset(cnt, 0, sizeof(cnt));
    }
    for (int i = 0; i < m; i++) cout << ans[i] << '\n';
}
```

树上莫队

将树上路径转化为欧拉序列后的区间问题

统计树上路径中不同的权值个数

```
int n, m, len, w[N];
int h[N], e[N], ne[N], idx;
int depth[N], fa[N][16];
int seq[N], top, first[N], last[N]; // 欧拉序列，每个点在树中第一次和最后一次出现的位置
int cnt[N], st[N], ans[N];
struct Query {
    int id, l, r, p; // p 表示询问中是否包含 lca
} q[N];
vector<int> nums;
void add_edge(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}
void dfs(int u, int fa) { // 求欧拉序列
    seq[++top] = u;
    first[u] = top;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (j != fa) dfs(j, u);
    }
    seq[++top] = u;
    last[u] = top;
}
void bfs() {
    memset(depth, 0x3f, sizeof(depth));
    depth[0] = 0, depth[1] = 1;
    queue<int> q;
    q.push(1);
    while (q.size()) {
        auto k = q.front();
        q.pop();
        for (int i = h[k]; i != -1; i = ne[i]) {
            int j = e[i];
            if (depth[j] > depth[k] + 1) {
                depth[j] = depth[k] + 1;
                q.push(j);
                fa[j][0] = k;
                for (int t = 1; t <= 15; t++) {
                    fa[j][t] = fa[fa[j][t - 1]][t - 1];
                }
            }
        }
    }
```

```
}
int lca(int a, int b) {
    if (depth[a] < depth[b]) swap(a, b);
    for (int k = 15; k >= 0; k--) {
        if (depth[fa[a][k]] >= depth[b]) a = fa[a][k];
    }
    if (a == b) return a;
    for (int k = 15; k >= 0; k--) {
        if (fa[a][k] != fa[b][k]) {
            a = fa[a][k];
            b = fa[b][k];
        }
    }
    return fa[a][0];
}
int get(int x) {
    return x / len;
}
bool cmp(Query a, Query b) {
    int i = get(a.l), j = get(b.l);
    if (i != j) return i < j;
    else return a.r < b.r;
}
void add(int x, int &res) {
    st[x] ^= 1;
    if (!st[x]) {
        cnt[w[x]]--;
        if (!cnt[w[x]]) res--;
    }
    else {
        if (!cnt[w[x]]) res++;
        cnt[w[x]]++;
    }
}
void solve() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        cin >> w[i];
        nums.push_back(w[i]);
    }
    sort(nums.begin(), nums.end());
    nums.erase(unique(nums.begin(), nums.end()), nums.end());
    for (int i = 1; i <= n; i++) w[i] = lower_bound(nums.begin(), nums.end(), w[i]) - nums.begin();
    memset(h, -1, sizeof(h));
    for (int i = 0; i < n - 1; i++) {
        int a, b;
        cin >> a >> b;
        add_edge(a, b), add_edge(b, a);
    }
    dfs(1, -1); // 求欧拉序列
    bfs();
    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        if (first[a] > first[b]) swap(a, b);
        int p = lca(a, b);
        if (a == p) q[i] = {i, first[a], first[b]};
        else q[i] = {i, last[a], first[b], p};
    }
    len = sqrt(top) + 1;
    sort(q, q + m, cmp);
    int res = 0;
    for (int i = 0, L = 1, R = 0; i < m; i++) {
        int id = q[i].id, l = q[i].l, r = q[i].r, p = q[i].p;
        while (R < r) add(seq[++R], res);
        while (R > r) add(seq[R--], res);
        while (L < l) add(seq[L++], res);
        while (L > l) add(seq[--L], res);
        if (p) add(p, res);
        ans[id] = res;
        if (p) add(p, res);
    }
    for (int i = 0; i < m; i++) cout << ans[i] << '\n';
}
```

二次离线莫队

题目看起来很适合使用莫队算法处理，但是他们的单次转移并非是 O(1) 的，这时直接使用莫队即使调整块长，也会导致复杂度不正确。此时，如果每次转移对答案的贡献可以进行差分，我们就可以将这些转移拆开离线下来，使用其它算法批量处理。我们用 f(x, l, r) 表示 x 关于 [l, r] 产生的贡献。如我们在将当前区间 [l, r] 扩展到 [l, r + 1] 时，我们要求的是 f(a*{r + 1}, l, r)。如果可以差分，我们可以将其写成 f(a*{r + 1}, 1, r) - f(a_{r + 1}, 1, l - 1)，其中第一项我们可以对于每个 r 都预处理出来，后一项我们可以把每个这样的项都离线存到对应的 l - 1 上，然后从小到大枚举并扫描线处理。其它几个转移的方向也都可以类似地处理。

每次查询一个区间内有多少对 (i, j) 满足 l <= i < j <= r 且 w[i] ^ w[j] 在二进制下有 k 个一

```
int n, m, k, len;
int w[N];
LL ans[N];
struct Query {
    int id, l, r;
    LL res;
} q[N];
struct Range {
    int id, l, r, t; // t 等于 -1 代表减去，等于 1 代表加上
};
vector<Range> range[N];
int f[N], g[N];
int get_count(int x) {
    int res = 0;
    while (x) {
        res += x & 1;
        x >>= 1;
    }
    return res;
}
int get(int x) {
    return x / len;
}
bool cmp(Query a, Query b) {
    int i = get(a.l), j = get(b.l);
    if (i != j) return i < j;
    else return a.r < b.r;
}
void solve() {
    cin >> n >> m >> k;
    for (int i = 1; i <= n; i++) cin >> w[i];
    vector<int> nums;
    for (int i = 0; i < 1 << 14; i++) {
        if (get_count(i) == k) nums.push_back(i);
    }
    for (int i = 1; i <= n; i++) {
        for (auto y : nums) g[w[i] ^ y]++;
        f[i] = g[w[i + 1]];
    }
    for (int i = 0; i < m; i++) {
        int l, r;
        cin >> l >> r;
        q[i] = {i, l, r};
    }
    len = sqrt(n) + 1;
    sort(q, q + m, cmp);
    for (int i = 0, L = 1, R = 0; i < m; i++) {
        int id = q[i].id, l = q[i].l, r = q[i].r;
        if (R < r) range[L - 1].push_back({i, R + 1, r, -1});
        while (R < r)  q[i].res += f[R++];
        if (R > r) range[L - 1].push_back({i, r + 1, R, 1});
        while (R > r) q[i].res -= f[--R];
        if (L < l) range[R].push_back({i, L, l - 1, -1});
        while (L < l) q[i].res += f[L - 1] + !k, L++;
        if (L > l) range[R].push_back({i, l, L - 1, 1});
        while (L > l) q[i].res -= f[L - 2] + !k, L--;
    }
    memset(g, 0, sizeof(g));
    for (int i = 1; i <= n; i++) {
        for (auto y: nums) g[w[i] ^ y]++;
        for (auto& rg: range[i]) {
            int id = rg.id, l = rg.l, r = rg.r, t = rg.t;
            for (int x = l; x <= r; x++) q[id].res += g[w[x]] * t;
        }
    }
    for (int i = 1; i < m; i++) q[i].res += q[i - 1].res;
    for (int i = 0; i < m; i++) ans[q[i].id] = q[i].res;
    for (int i = 0; i < m; i++) cout << ans[i] << '\n';
}
```

树链剖分

将一棵树转化成一个序列, 将一个路径转化为 log 段的区间

支持路径加，路径求和，子树加，子树求和

```
int n, m, root; // n 个点 m 个操作根节点为 root
int w[N], id[N], nw[N], cnt; // w 点权 nw 新的权值
int dep[N], sz[N], top[N], fa[N], son[N];
vector<int> adj[N];
struct Tree {
    int l, r;
    i64 add, sum;
} tr[N * 4];
void dfs1(int u, int father, int depth) { // 更新 dep, fa, sz, son 的信息
    dep[u] = depth, fa[u] = father, sz[u] = 1;
    for (auto v : adj[u]) {
        if (v == father) continue;
        dfs1(v, u, depth + 1);
        sz[u] += sz[v];
        if (sz[son[u]] < sz[v]) son[u] = v;
    }
}
void dfs2(int u, int t) { // t 是当前点所在重链的顶点
    id[u] = ++cnt, nw[cnt] = w[u], top[u] = t;
    if (!son[u]) return;
    dfs2(son[u], t);
    for (auto v : adj[u]) {
        if (v == fa[u] || v == son[u]) continue;
        dfs2(v, v);
    }
}
void pushup(int u) {
    tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
}
void pushdown(int u) {
    auto &root = tr[u], &le = tr[u << 1], &ri = tr[u << 1 | 1];
    if (root.add) {
        le.add += root.add, le.sum += root.add * (le.r - le.l + 1);
        ri.add += root.add, ri.sum += root.add * (ri.r - ri.l + 1);
        root.add = 0;
    }
}
void build(int u, int l, int r) {
    tr[u] = {l, r, 0, nw[r]};
    if (l == r) return;
    int mid = l + r >> 1;
    build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
    pushup(u);
}
void update(int u, int l, int r, int k) {
    if (l <= tr[u].l && r >= tr[u].r) {
        tr[u].add += k;
        tr[u].sum += (i64)k * (tr[u].r - tr[u].l + 1);
        return;
    }
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    if (l <= mid) update(u << 1, l, r, k);
    if (r > mid) update(u << 1 | 1, l, r, k);
    pushup(u);
}
i64 query(int u, int l, int r) {
    if (l <= tr[u].l && r >= tr[u].r) return tr[u].sum;
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    i64 res = 0;
    if (l <= mid) res += query(u << 1, l, r);
    if (r > mid) res += query(u << 1 | 1, l, r);
    return res;
}
void update_path(int u, int v, int k) {
    while (top[u] != top[v]) {
        if (dep[top[u]] < dep[top[v]]) swap(u, v);
        update(1, id[top[u]], id[u], k);
        u = fa[top[u]];
    }
    if (dep[u] < dep[v]) swap(u, v); // id[v] 为原始 (u,v) 的 lca
    update(1, id[v], id[u], k);
}
i64 query_path(int u, int v) {
    i64 res = 0;
```

```
    while (top[u] != top[v]) {
        if (dep[top[u]] < dep[top[v]]) swap(u, v);
        res += query(1, id[top[u]], id[u]);
        u = fa[top[u]];
    }
    if (dep[u] < dep[v]) swap(u, v);
    res += query(1, id[v], id[u]);
    return res;
}
void update_tree(int u, int k) {
    update(1, id[u], id[u] + sz[u] - 1, k);
}
i64 query_tree(int u) {
    return query(1, id[u], id[u] + sz[u] - 1);
}
void solve() {
    int p;
    cin >> n >> m >> root;
    for (int i = 1; i <= n; i++) cin >> w[i];
    for (int i = 0; i < n - 1; i++) {
        int a, b;
        cin >> a >> b;
        adj[a].push_back(b), adj[b].push_back(a);
    }
    dfs1(root, 0, 1);
    dfs2(root, root);
    build(1, 1, n);
    while (m--) {
        int op, u, v, k;
        cin >> op;
        if (op == 1) { // 路径加 k
            cin >> u >> v >> k;
            update_path(u, v, k);
        }
        else if (op == 2) { // 输出路径权值和
            cin >> u >> v;
            cout << query_path(u, v) << '\n';
        }
        else if (op == 3) { // 子树加 k
            cin >> u >> k;
            update_tree(u, k);
        }
        else {
            cin >> u;
            cout << query_tree(u) << '\n';
        }
    }
}
```

树套树

支持单点修改，查询一个区间内的前驱

```
int n, m;
struct Tree {
    int l, r;
    multiset<int> s;
} tr[N * 4];
int w[N];
void build(int u, int l, int r) {
    tr[u] = {l, r};
    tr[u].s.insert(INF), tr[u].s.insert(-INF);
    for (int i = l; i <= r; i++) tr[u].s.insert(w[i]);
    if (l == r) return;
    int mid = l + r >> 1;
    build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
}
void change(int u, int p, int x) {
    tr[u].s.erase(tr[u].s.find(w[p]));
    tr[u].s.insert(x);
    if (tr[u].l == tr[u].r) return;
    int mid = tr[u].l + tr[u].r >> 1;
    if (p <= mid) change(u << 1, p, x);
    else change(u << 1 | 1, p, x);
}
int query(int u, int a, int b, int x) {
    if (tr[u].l >= a && tr[u].r <= b) {
        auto it = tr[u].s.lower_bound(x);
        it--;
        return *it;
```

```
    }
    int mid = tr[u].l + tr[u].r >> 1;
    int res = -INF;
    if (a <= mid) res = max(res, query(u << 1, a, b, x));
    if (b > mid) res = max(res, query(u << 1 | 1, a, b, x));
    return res;
}
void solve() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> w[i];
    build(1, 1, n);
    while (m--) {
        int op;
        cin >> op;
        if (op == 1) { // 修改一个点
            int pos, x;
            cin >> pos >> x;
            change(1, pos, x);
            w[pos] = x;
        }
        else { // 查询区间中的前驱
            int l, r, x;
            cin >> l >> r >> x;
            cout << query(1, l, r, x) << '\n';
        }
    }
}
```

1.查询 k 在区间的排名

2.查询区间排名为 k 的数

3.修改某一位置上的数

4.查询 k 在区间的前驱

5.查询 k 在区间的后继

```
int n, m;
struct Node {
    int s[2], v, p, sz;
    void init(int _v, int _p) {
        v = _v, p = _p;
        sz = 1;
    }
} tr[N];
int L[N], R[N], T[N], idx;
int w[N];
void pushup(int x) {
    tr[x].sz = tr[tr[x].s[0]].sz + tr[tr[x].s[1]].sz + 1;
}
void rotate(int x) {
    int y = tr[x].p, z = tr[y].p;
    int k = tr[y].s[1] == x;
    tr[z].s[tr[z].s[1] == y] = x, tr[x].p = z;
    tr[y].s[k] = tr[x].s[k ^ 1], tr[tr[x].s[k ^ 1]].p = y;
    tr[x].s[k ^ 1] = y, tr[y].p = x;
    pushup(y), pushup(x);
}
void splay(int& root, int x, int k) {
    while (tr[x].p != k) {
        int y = tr[x].p, z = tr[y].p;
        if (z != k) {
            if ((tr[y].s[1] == x) ^ (tr[z].s[1] == y)) rotate(x);
            else rotate(y);
        }
        rotate(x);
    }
    if (!k) root = x;
}
void insert(int& root, int v) {
    int u = root, p = 0;
    while (u) p = u, u = tr[u].s[v > tr[u].v];
    u = ++ idx;
    if (p) tr[p].s[v > tr[p].v] = u;
    tr[u].init(v, p);
    splay(root, u, 0);
}
int get_k(int root, int v) {
    int u = root, res = 0;
```

```
    while (u) {
        if (tr[u].v < v) res += tr[tr[u].s[0]].sz + 1, u = tr[u].s[1];
        else u = tr[u].s[0];
    }
    return res;
}
void update(int& root, int x, int y) {
    int u = root;
    while (u) {
        if (tr[u].v == x) break;
        if (tr[u].v < x) u = tr[u].s[1];
        else u = tr[u].s[0];
    }
    splay(root, u, 0);
    int l = tr[u].s[0], r = tr[u].s[1];
    while (tr[l].s[1]) l = tr[l].s[1];
    while (tr[r].s[0]) r = tr[r].s[0];
    splay(root, l, 0), splay(root, r, l);
    tr[r].s[0] = 0;
    pushup(r), pushup(l);
    insert(root, y);
}
void build(int u, int l, int r) {
    L[u] = l, R[u] = r;
    insert(T[u], -INF), insert(T[u], INF);
    for (int i = l; i <= r; i++) insert(T[u], w[i]);
    if (l == r) return;
    int mid = l + r >> 1;
    build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
}
int query(int u, int a, int b, int x) {
    if (L[u] >= a && R[u] <= b) return get_k(T[u], x) - 1;
    int mid = L[u] + R[u] >> 1;
    int res = 0;
    if (a <= mid) res += query(u << 1, a, b, x);
    if (b > mid) res += query(u << 1 | 1, a, b, x);
    return res;
}
void change(int u, int p, int x) {
    update(T[u], w[p], x);
    if (L[u] == R[u]) return;
    int mid = L[u] + R[u] >> 1;
    if (p <= mid) change(u << 1, p, x);
    else change(u << 1 | 1, p, x);
}
int get_pre(int root, int v) {
    int u = root, res = -INF;
    while (u) {
        if (tr[u].v < v) res = max(res, tr[u].v), u = tr[u].s[1];
        else u = tr[u].s[0];
    }
    return res;
}
int get_suc(int root, int v) {
    int u = root, res = INF;
    while (u) {
        if (tr[u].v > v) res = min(res, tr[u].v), u = tr[u].s[0];
        else u = tr[u].s[1];
    }
    return res;
}
int query_pre(int u, int a, int b, int x) {
    if (L[u] >= a && R[u] <= b) return get_pre(T[u], x);
    int mid = L[u] + R[u] >> 1;
    int res = -INF;
    if (a <= mid) res = max(res, query_pre(u << 1, a, b, x));
    if (b > mid) res = max(res, query_pre(u << 1 | 1, a, b, x));
    return res;
}
int query_suc(int u, int a, int b, int x) {
    if (L[u] >= a && R[u] <= b) return get_suc(T[u], x);
    int mid = L[u] + R[u] >> 1;
    int res = INF;
    if (a <= mid) res = min(res, query_suc(u << 1, a, b, x));
    if (b > mid) res = min(res, query_suc(u << 1 | 1, a, b, x));
    return res;
}
void solve() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> w[i];
    build(1, 1, n);
```

```
    while (m--) {
        int op;
        cin >> op;
        if (op == 1) {
            int l, r, x;
            cin >> l >> r >> x;
            cout << query(1, l, r, x) + 1 << '\n';
        }
        else if (op == 2) {
            int a, b, k;
            cin >> a >> b >> k;
            int l = 0, r = 1e8;
            while (l < r) {
                int mid = l + r + 1 >> 1;
                if (query(1, a, b, mid) + 1 <= k) l = mid;
                else r = mid - 1;
            }
            cout << r << '\n';
        }
        else if (op == 3) {
            int pos, x;
            cin >> pos >> x;
            change(1, pos, x);
            w[pos] = x;
        }
        else if (op == 4) {
            int l, r, x;
            cin >> l >> r >> x;
            int ans = query_pre(1, l, r, x);
            if (ans == -INF) cout << -2147483647 << '\n';
            else cout << ans << '\n';
        }
        else {
            int l, r, x;
            cin >> l >> r >> x;
            int ans = query_suc(1, l, r, x);
            if (ans == INF) cout << 2147483647 << '\n';
            else cout << ans << '\n';
        }
    }
}
```

树状数组

```
int a[N];
long long tr[N];
int lowbit(int x) {
    return x & -x;
}
void add(int x, long long k) {    // 单点修改
    for (int i = x; i <= n; i += lowbit(i)) {
        tr[i] += k;
    }
}
long long sum(int x) {         // 求前缀和
    long long res = 0;
    for (int i = x; i; i -= lowbit(i)) {
        res += tr[i];
    }
    return res;
}
```

线段树

单点修改，区间查询，不需懒标记

求区间最大连续子段和

```
int n, m;
int w[N];
struct Node { // 求区间最大连续子段和
    int l, r;
    int sum, lmax, rmax, tmax;
} tr[N * 4];
void pushup(Node &u, Node &l, Node &r) {
    u.sum = l.sum + r.sum;
    u.lmax = max(l.lmax, l.sum + r.lmax);
    u.rmax = max(r.rmax, r.sum + l.rmax);
    u.tmax = max({l.tmax, r.tmax, l.rmax + r.lmax});
```

```
}
void pushup(int u) { // 由子节点的信息计算父节点的信息
    pushup(tr[u], tr[u << 1], tr[u << 1 | 1]);
}
void build(int u, int l, int r) {
    if (l == r) tr[u] = {l, r, w[r], w[r], w[r], w[r]};
    else {
        tr[u] = {l, r};
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
        pushup(u);
    }
}
Node query(int u, int l, int r) {
    if (tr[u].l >= l && tr[u].r <= r) return tr[u]; // 树中节点完全包含在[l, r]中
    int mid = tr[u].l + tr[u].r >> 1;
    if (r <= mid) return query(u << 1, l, r);
    else if (l > mid) return query(u << 1 | 1, l, r);
    else {
        auto le = query(u << 1, l, r);
        auto ri = query(u << 1 | 1, l, r);
        Node res;
        pushup(res, le, ri);
        return res;
    }
}
void modify(int u, int x, int v) {
    if (tr[u].l == x && tr[u].r == x) tr[u] = {x, x, v, v, v, v};
    else {
        int mid = tr[u].l + tr[u].r >> 1;
        if (x <= mid) modify(u << 1, x, v);
        else modify(u << 1 | 1, x, v);
        pushup(u);
    }
}
void solve() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> w[i];
    build(1, 1, n);
    int k, x, y;
    while (m--) {
        cin >> k >> x >> y;
        if (k == 1) {
            if (x > y) swap(x, y);
            cout << query(1, x, y).tmax << '\n';
        }
        else modify(1, x, y);
    }
}
```

区间最大公约数 支持区间加，区间查询

```
int n, m;
LL w[N];
struct Node {
    int l, r;
    LL sum, d;
} tr[N * 4];
void pushup(Node &u, Node &l, Node &r) {
    u.sum = l.sum + r.sum;
    u.d = __gcd(l.d, r.d);
}
void pushup(int u) {
    pushup(tr[u], tr[u << 1], tr[u << 1 | 1]);
}
void build(int u, int l, int r) {
    if (l == r) {
        LL b = w[r] - w[r - 1];
        tr[u] = {l, r, b, b};
    }
    else {
        tr[u] = {l, r};
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
        pushup(u);
    }
}
void modify(int u, int x, LL v) {
    if (tr[u].l == x && tr[u].r == x) {
        LL b = tr[u].sum + v;
```

```
        tr[u] = {x, x, b, b};
    }
    else {
        int mid = tr[u].l + tr[u].r >> 1;
        if (x <= mid) modify(u << 1, x, v);
        else modify(u << 1 | 1, x, v);
        pushup(u);
    }
}
Node query(int u, int l, int r) {
    if (l > r) return {0};
    if (tr[u].l >= l && tr[u].r <= r) return tr[u];
    else {
        int mid = tr[u].l + tr[u].r >> 1;
        if (r <= mid) return query(u << 1, l, r);
        else if (l > mid) return query(u << 1 | 1, l, r);
        else {
            auto le = query(u << 1, l, r);
            auto ri = query(u << 1 | 1, l, r);
            Node res;
            pushup(res, le, ri);
            return res;
        }
    }
}
void solve() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> w[i];
    build(1, 1, n);
    int l, r;
    LL d;
    string op;
    while (m--) {
        cin >> op >> l >> r;
        if (op[0] == 'Q') {
            auto le = query(1, 1, l);
            auto ri = query(1, l + 1, r);
            cout << abs(__gcd(le.sum, ri.d)) << '\n';
        }
        else {
            cin >> d;
            modify(1, l, d);
            if (r + 1 <= n) modify(1, r + 1, -d);
        }
    }
}
```

区间加，区间求和

```
int n, m;
int w[N];
struct Node {
    int l, r;
    i64 sum, add;
} tr[N * 4];
void pushup(int u) {
    tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
}
void pushdown(int u) {
    auto &root = tr[u], &le = tr[u << 1], &ri = tr[u << 1 | 1];
    if (root.add) {
        le.add += root.add, le.sum += i64(le.r - le.l + 1) * root.add;
        ri.add += root.add, ri.sum += i64(ri.r - ri.l + 1) * root.add;
        root.add = 0;
    }
}
void build(int u, int l, int r) {
    if (l == r) tr[u] = {l, l, w[l], 0};
    else {
        tr[u] = {l, r};
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
        pushup(u);
    }
}
void modify(int u, int l, int r, i64 v) {
    if (tr[u].l >= l && tr[u].r <= r) {
        tr[u].sum += LL(tr[u].r - tr[u].l + 1) * v;
        tr[u].add += v;
    }
```

```
    else { // 一定要分裂
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid) modify(u << 1, l, r, v);
        if (r > mid) modify(u << 1 | 1, l, r, v);
        pushup(u);
    }
}
i64 query(int u, int l, int r) {
    if (tr[u].l >= l && tr[u].r <= r) return tr[u].sum;
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    i64 sum = 0;
    if (l <= mid) sum = query(u << 1, l, r);
    if (r > mid) sum += query(u << 1 | 1, l, r);
    return sum;
}
void solve() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> w[i];
    build(1, 1, n);
    string op;
    int l, r;
    i64 v;
    while (m--) {
        cin >> op >> l >> r;
        if (op[0] == 'C') {
            cin >> v;
            modify(1, l, r, v);
        }
        else {
            cout << query(1, l, r) << '\n';
        }
    }
}
```

区间加, 区间极值

```
int n;
struct Node {
    int l, r;
    int add, low, lownum; // 区间极小值和极小值个数
} tr[N * 4];
void pushup(int u) {
    if (tr[u << 1].low < tr[u << 1 | 1].low) tr[u].low = tr[u << 1].low, tr[u].lownum = tr[u << 1].lownum;
    else if (tr[u << 1].low > tr[u << 1 | 1].low) tr[u].low = tr[u << 1 | 1].low, tr[u].lownum = tr[u << 1 | 1].lownum;
    else {
        tr[u].low = tr[u << 1].low;
        tr[u].lownum = tr[u << 1].lownum + tr[u << 1 | 1].lownum;
    }
}
void pushdown(int u) {
    auto &root = tr[u], &le = tr[u << 1], &ri = tr[u << 1 | 1];
    if (root.add) {
        le.add += root.add, le.low += root.add;
        ri.add += root.add, ri.low += root.add;
        root.add = 0;
    }
}
void build(int u, int l, int r) {
    if (l == r) tr[u] = {l, l, 0, 0, 1};
    else {
        tr[u] = {l, r, 0};
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
        pushup(u);
    }
}
void modify(int u, int l, int r, int v) {
    if (tr[u].l >= l && tr[u].r <= r) {
        tr[u].low += v;
        tr[u].add += v;
    }
    else { // 一定要分裂
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid) modify(u << 1, l, r, v);
        if (r > mid) modify(u << 1 | 1, l, r, v);
        pushup(u);
    }
```

```
}
LL query(int u, int l, int r) {
    if (tr[u].l >= l && tr[u].r <= r) {
        if (tr[u].low == 0) return tr[u].lownum;
        else return 0;
    }
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    LL sum = 0;
    if (l <= mid) sum = query(u << 1, l, r);
    if (r > mid) sum += query(u << 1 | 1, l, r);
    return sum;
}
```

区间加，区间乘，区间求和

```
int n, m, p; // m 个查询，模数为 p
int w[N];
struct Node {
    int l, r;
    int sum, add, mul;
} tr[N * 4];
void pushup(int u) {
    tr[u].sum = (tr[u << 1].sum + tr[u << 1 | 1].sum) % p;
}
void eval(Node &t, int add, int mul) {
    t.sum = ((LL)t.sum * mul + (LL)add * (t.r - t.l + 1)) % p;
    t.mul = (LL)t.mul * mul % p;
    t.add = ((LL)t.add * mul + add) % p;
}
void pushdown(int u) {
    eval(tr[u << 1], tr[u].add, tr[u].mul);
    eval(tr[u << 1 | 1], tr[u].add, tr[u].mul);
    tr[u].add = 0, tr[u].mul = 1;
}
void build(int u, int l, int r) {
    if (l == r) tr[u] = {l, r, w[l], 0, 1};
    else {
        tr[u] = {l, r, 0, 0, 1};
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
        pushup(u);
    }
}
void modify(int u, int l, int r, int add, int mul) {
    if (tr[u].l >= l && tr[u].r <= r) eval(tr[u], add, mul);
    else {
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid) modify(u << 1, l, r, add, mul);
        if (r > mid) modify(u << 1 | 1, l, r, add, mul);
        pushup(u);
    }
}
int query(int u, int l, int r) {
    if (tr[u].l >= l && tr[u].r <= r) return tr[u].sum;
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    int sum = 0;
    if (l <= mid) sum = query(u << 1, l, r);
    if (r > mid) sum = (sum + query(u << 1 | 1, l, r)) % p;
    return sum;
}
void solve() {
    cin >> n >> p;
    for (int i = 1; i <= n; i++) cin >> w[i];
    build(1, 1, n);
    cin >> m;
    while (m--) {
        int t, l, r, d;
        cin >> t >> l >> r;
        if (t == 1) {
            cin >> d;
            modify(1, l, r, 0, d);
        }
        else if (t == 2) {
            cin >> d;
            modify(1, l, r, d, 1);
        }
        else cout << query(1, l, r) << '\n';
```

```
    }
}
```

压缩Trie树

```
// 构建压缩前缀树
void buildTrie(int u, int l, int r, int depth) {
    if (l == r) {
        leafPos[arr[l].id] = u;
        return;
    }
    int startDepth = depth;
    while (depth < (int)arr[l].s.size()
           && depth < (int)arr[r].s.size()
           && arr[l].s[depth] == arr[r].s[depth]) {
        ++depth;
    }
    // 如果有公共前缀，就新建中间节点
    if (depth > startDepth) {
        int v = ++nodeCnt;
        tree[u].children.push_back(v);
        tree[v].parent = u;
        u = v;
    }
    // 按第 depth 位分组递归
    for (int i = l, j; i <= r; i = j) {
        j = i + 1;
        while (j <= r && arr[j].s[depth] == arr[i].s[depth]) {
            ++j;
        }
        int v = ++nodeCnt;
        tree[u].children.push_back(v);
        tree[v].parent = u;
        buildTrie(v, i, j - 1, depth + 1);
    }
}
```

左偏树、可并堆

对于一棵二叉树，我们定义 外节点 为子节点数小于两个的节点，定义一个节点的 dist 为其到子树中最近的外节点所经过的边的数量。空节点的 dist 为 0。左偏树是一棵二叉树，它不仅具有堆的性质，并且是「左偏」的：每个节点左儿子的 dist 都大于等于右儿子的 dist。因此，左偏树每个节点的 dist 都等于其右儿子的 dist 加一。

删除根 合并根的左右儿子即可。

删除任意节点 先将左右儿子合并，然后自底向上更新 dist、不满足左偏性质时交换左右儿子，当 dist 无需更新时结束递归：

整个堆加上/减去一个值、乘上一个正数 在根打上标记，删除根/合并堆（访问儿子）时下传标记即可

```
int n;
int v[N], dist[N], l[N], r[N], idx; // 权值，距离，左儿子，右儿子
int p[N]; // 并查集
bool st[N];
int find(int x) {
    if (p[x] != x) p[x] = find(p[x]);
    return p[x];
}
bool cmp(int x, int y) { // 判断 x 是否小于 y
    if (v[x] != v[y]) return v[x] < v[y];
    return x < y;
}
int merge(int x, int y) {
    if (!x || !y) return x + y;
    if (cmp(y, x)) swap(x, y);
    r[x] = merge(r[x], y);
    if (dist[r[x]] > dist[l[x]]) swap(r[x], l[x]);
    dist[x] = dist[r[x]] + 1;
    return x;
}
void solve() {
    cin >> n;
    while (n--) {
        int t;
        cin >> t;
        if (t == 1) { // 新建左偏树堆
            int x;
            cin >> x;
            v[++idx] = x;
```

```
            dist[idx] = 1;
            p[idx] = idx;
        }
        else if (t == 2) { // 将第 x 个插入的数和第 y 个插入的数所在的左偏树合并(如果 x 或 y 被删过了或者 x 和 y 在同一个左偏树上就忽视)
            int x, y;
            cin >> x >> y;
            if (!st[x] && !st[y]) {
                x = find(x), y = find(y);
                if (x != y) {
                    if (cmp(y, x)) swap(x, y);
                    p[y] = x;
                    merge(x, y);
                }
            }
        }
        else if (t == 3) { // 输出第 x 个数所在左偏树的堆顶
            int x;
            cin >> x;
            x = find(x);
            cout << v[x] << '\n';
        }
        else { // 删除第 x 个数所在左偏树的最值（如果相同的有多个删除最早插入的数）如果第 x 个数已经被删除输出 -1
            int x;
            cin >> x;
            if (st[x]) {
                cout << -1 << '\n';
                continue;
            }
            x = find(x);
            st[x] = 1;
            int nowtop = merge(l[x], r[x]);
            p[nowtop] = nowtop, p[x] = nowtop;
        }
    }
}
```

CDQ分治

三维偏序

nlog^2 每个点有三个属性，试求：这个序列里有多少对点对 (i, j) 满足

$$\text{Count} = \left|\{(i,j) \mid a_j \le a_i,\ b_j \le b_i,\ c_j \le c_i,\ j \ne i\}\right| \tag{1}$$

```
int n, m;
struct Data {
    int a, b, c, s, res;   // s 是当前点出现的次数，res 是满足三维偏序的个数
    bool operator< (const Data& t) const {
        if (a != t.a) return a < t.a;
        if (b != t.b) return b < t.b;
        return c < t.c;
    }
    bool operator== (const Data& t) {
        return (a == t.a && b == t.b && c == t.c);
    }
} q[N], w[N]; // w 是归并排序的辅助数组
int tr[M], ans[N];
int lowbit(int x) {
    return x & -x;
}
void add(int x, int v) {
    for (int i = x; i < M; i += lowbit(i)) tr[i] += v;
}
int query(int x) {
    int res = 0;
    for (int i = x; i; i -= lowbit(i)) res += tr[i];
    return res;
}
void merge_sort(int l, int r) {
    if (l >= r) return;
    int mid = l + r >> 1;
    merge_sort(l, mid), merge_sort(mid + 1, r);
    int i = l, j = mid + 1, k = 0;
    while (i <= mid && j <= r) {
        if (q[i].b <= q[j].b) add(q[i].c, q[i].s), w[k++] = q[i++];
        else q[j].res += query(q[j].c), w[k++] = q[j++];
    }
    while (i <= mid) add(q[i].c, q[i].s), w[k++] = q[i++];
    while (j <= r) q[j].res += query(q[j].c), w[k++] = q[j++];
```

```
    for (i = l; i <= mid; i++) add(q[i].c, -q[i].s);
    for (i = l, j = 0; j < k; i++, j++) q[i] = w[j];
}
void solve() {
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        q[i] = {a, b, c, 1};
    }
    sort(q, q + n);
    int k = 1;
    for (int i = 1; i < n; i++) {
        if (q[i] == q[k - 1]) q[k - 1].s++;
        else q[k++] = q[i];
    }
    merge_sort(0, k - 1);
    for (int i = 0; i < k; i++) ans[q[i].res + q[i].s - 1] += q[i].s;
    for (int i = 0; i < n; i++) cout << ans[i] << '\n';
}
```

二维区域和、二维数点

```
int n, m;
struct Data {
    int x, y, z, p, id, sign; // z 表示是查询点还是真实点, p 是点权值, id 是每个点属于哪个询问, sign 表示正负
    LL sum;
    bool operator< (const Data& t) const {
        if (x != t.x) return x < t.x;
        if (y != t.y) return y < t.y;
        return z < t.z;
    }
} q[N], w[N];
LL ans[N];
void merge_sort(int l, int r) {
    if (l >= r) return;
    int mid = l + r >> 1;
    merge_sort(l, mid), merge_sort(mid + 1, r);
    int i = l, j = mid + 1, k = 0;
    LL sum = 0;
    while (i <= mid && j <= r) {
        if (q[i].y <= q[j].y) sum += !q[i].z * q[i].p, w[k++] = q[i++];
        else q[j].sum += sum, w[k++] = q[j++];
    }
    while (i <= mid) sum += !q[i].z * q[i].p, w[k++] = q[i++];
    while (j <= r) q[j].sum += sum, w[k++] = q[j++];
    for (i = l, j = 0; j < k; i++, j++) q[i] = w[j];
}
void solve() {
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        int x, y, p;
        cin >> x >> y >> p;
        q[i] = {x, y, 0, p};
    }
    int k = n;
    for (int i = 1; i <= m; i++) {
        int x, y, xx, yy;
        cin >> x >> y >> xx >> yy;
        q[k++] = {xx, yy, 1, 0, i, 1};
        q[k++] = {x - 1, yy, 1, 0, i, -1};
        q[k++] = {xx, y - 1, 1, 0, i, -1};
        q[k++] = {x - 1, y - 1, 1, 0, i, 1};
    }
    sort(q, q + k);
    merge_sort(0, k - 1);
    for (int i = 0; i < k; i++) {
        if (q[i].z) {
            ans[q[i].id] += q[i].sum * q[i].sign;
        }
    }
    for (int i = 1; i <= m; i++) cout << ans[i] << '\n';
}
```

动态逆序对

现在给出 1~n 的一个排列，按照某种顺序依次删除 m 个元素，你的任务是在每次删除一个元素之前统计整个序列的逆序对数。

```
int n, m;
struct Data {
```

```
    int a, t, res; // 值，时间戳，当前点的逆序对数
} q[N], w[N];
int tr[N], pos[N]; // pos 是值对下标的映射
LL ans[N];
int lowbit(int x) {
    return x & -x;
}
void add(int x, int v) {
    for (int i = x; i < N; i += lowbit(i)) tr[i] += v;
}
LL query(int x) {
    LL res = 0;
    for (int i = x; i; i -= lowbit(i)) res += tr[i];
    return res;
}
void merge_sort(int l, int r) {
    if (l >= r) return;
    int mid = l + r >> 1;
    merge_sort(l, mid), merge_sort(mid + 1, r);
    int i = mid, j = r;
    while (i >= l && j >= mid + 1) {
        if (q[i].a > q[j].a) add(q[i].t, 1), i--;
        else q[j].res += query(q[j].t - 1), j--;
    }
    while (j >= mid + 1) q[j].res += query(q[j].t - 1), j--;
    for (int k = i + 1; k <= mid; k++) add(q[k].t, -1);
    j = l, i = mid + 1;
    while (j <= mid && i <= r) {
        if (q[i].a < q[j].a) add(q[i].t, 1), i++;
        else q[j].res += query(q[j].t - 1), j++;
    }
    while (j <= mid) q[j].res += query(q[j].t - 1), j++;
    for (int k = mid + 1; k < i; k++) add(q[k].t, -1);
    i = l, j = mid + 1;
    int k = 0;
    while (i <= mid && j <= r) {
        if (q[i].a <= q[j].a) w[k++] = q[i++];
        else w[k++] = q[j++];
    }
    while (i <= mid) w[k++] = q[i++];
    while (j <= r) w[k++] = q[j++];

    for (i = l, j = 0; j < k; i++, j++) q[i] = w[j];
}
void solve() {
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        cin >> q[i].a;
        pos[q[i].a] = i;
    }
    for (int i = 0, j = n; i < m; i++) {
        int a;
        cin >> a;
        q[pos[a]].t = j--;
        pos[a] = -1;
    }
    for (int i = 1, j = n - m; i <= n; i++) {
        if (pos[i] != -1) {
            q[pos[i]].t = j--;
        }
    }
    merge_sort(0, n - 1);
    for (int i = 0; i < n; i++) ans[q[i].t] = q[i].res;
    for (int i = 2; i <= n; i++) ans[i] += ans[i - 1];
    for (int i = 0, j = n; i < m; i++, j--) cout << ans[j] << '\n';
}
```

DSU on tree

求每个子树上颜色出现次数最多的，所有颜色之和

```
int n;
int h[N], e[M], ne[M], idx;
int color[N], cnt[N], sz[N], son[N];
LL ans[N], sum;
int mx; // 颜色出现最大次数
void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}
int dfs_son(int u, int father) { // 确定重儿子
```

```
    sz[u] = 1;
    for (int i = h[u]; ~i; i = ne[i]) {
        int j = e[i];
        if (j == father) continue;
        sz[u] += dfs_son(j, u);
        if (sz[j] > sz[son[u]]) son[u] = j;
    }
    return sz[u];
}
void update(int u, int father, int sign, int pson) { // sign 表示加或者清空
    int c = color[u];
    cnt[c] += sign;
    if (cnt[c] > mx) mx = cnt[c], sum = c;
    else if (cnt[c] == mx) sum += c;
    for (int i = h[u]; ~i; i = ne[i]) {
        int j = e[i];
        if (j == father || j == pson) continue;
        update(j, u, sign, pson);
    }
}
void dfs(int u, int father, int op) { // op == 0 为轻儿子不需保留，op == 1 为重儿子贡献保留
    for (int i = h[u]; ~i; i = ne[i]) {
        int j = e[i];
        if (j == father || j == son[u]) continue;
        dfs(j, u, 0);
    }
    if (son[u]) dfs(son[u], u, 1);
    update(u, father, 1, son[u]);
    ans[u] = sum;
    if (!op) update(u, father, -1, 0), sum = mx = 0;
}
void solve() {
    cin >> n;
    memset(h, -1, sizeof(h));
    for (int i = 1; i <= n; i ++ ) cin >> color[i];
    for (int i = 0; i < n - 1; i++) {
        int a, b;
        cin >> a >> b;
        add(a, b), add(b, a);
    }
    dfs_son(1, -1);
    dfs(1, -1, 1);
    for (int i = 1; i <= n; i++) cin >> ans[i];
}
```

LCT

动态查询两点之间是否连通可以通过判断原树根节点是否相同来判断, 当然用 lct 判断教慢，如果只有加边可以使用并查集

查询路径点权值异或和，动态加边，删边，修改点权

```
int n, m;
struct Node {
    int s[2], p, v;
    int sum, rev;
} tr[N];
int stk[N];
void pushrev(int x) {
    swap(tr[x].s[0], tr[x].s[1]);
    tr[x].rev ^= 1;
}
void pushup(int x) {
    tr[x].sum = tr[tr[x].s[0]].sum ^ tr[x].v ^ tr[tr[x].s[1]].sum;
}
void pushdown(int x) {
    if (tr[x].rev) {
        pushrev(tr[x].s[0]), pushrev(tr[x].s[1]);
        tr[x].rev = 0;
    }
}
bool isroot(int x) {
    return tr[tr[x].p].s[0] != x && tr[tr[x].p].s[1] != x;
}
void rotate(int x) {
    int y = tr[x].p, z = tr[y].p;
    int k = tr[y].s[1] == x;
    if (!isroot(y)) tr[z].s[tr[z].s[1] == y] = x;
    tr[x].p = z;
    tr[y].s[k] = tr[x].s[k ^ 1], tr[tr[x].s[k ^ 1]].p = y;
    tr[x].s[k ^ 1] = y, tr[y].p = x;
```

```
    pushup(y), pushup(x);
}
void splay(int x) {
    int top = 0, r = x;
    stk[++top] = r;
    while (!isroot(r)) stk[++top] = r = tr[r].p;
    while (top) pushdown(stk[top--]);
    while (!isroot(x)) {
        int y = tr[x].p, z = tr[y].p;
        if (!isroot(y)) {
            if ((tr[y].s[1] == x) ^ (tr[z].s[1] == y)) rotate(x);
            else rotate(y);
        }
        rotate(x);
    }
}
void access(int x) { // 建立一条从根到 x 的路径，同时将 x 变成 splay 的根节点
    int z = x;
    for (int y = 0; x; y = x, x = tr[x].p) {
        splay(x);
        tr[x].s[1] = y, pushup(x);
    }
    splay(z);
}
void makeroot(int x) { // 将 x 变成原树的根节点
    access(x);
    pushrev(x);
}
int findroot(int x) { // 找到 x 所在原树的根节点，再将原树的根节点旋转到 splay 的根节点
    access(x);
    while (tr[x].s[0]) pushdown(x), x = tr[x].s[0];
    splay(x);
    return x;
}
void split(int x, int y) { // 给 x 和 y 之间的路径建立一个 splay，其根节点是 y
    makeroot(x);
    access(y);
}
void link(int x, int y) { // 如果 x 和 y 不连通，则加入一条 x 和 y 之间的边
    makeroot(x);
    if (findroot(y) != x) tr[x].p = y;
}
void cut(int x, int y) { // 如果 x 和 y 之间存在边，则删除该边
    makeroot(x);
    if (findroot(y) == x && tr[y].p == x && !tr[y].s[0]) {
        tr[x].s[1] = tr[y].p = 0;
        pushup(x);
    }
}
void solve() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> tr[i].v;
    while (m--) {
        int t, x, y;
        cin >> t >> x >> y;
        if (t == 0) {
            split(x, y);
            cout << tr[y].sum << '\n';
        }
        else if (t == 1) link(x, y);
        else if (t == 2) cut(x, y);
        else {
            splay(x);
            tr[x].v = y;
            pushup(x);
        }
    }
}
```

Splay

区间翻转

多棵树时 root 换为 root[b]

```
int n, m;
struct Node {
    int s[2], p, v;
    int sz, flag;
    void init(int _v, int _p) {
```

```
        v = _v, p = _p;
        sz = 1;
    }
} tr[N];
int root, idx;
void pushup(int x) {
    tr[x].sz = tr[tr[x].s[0]].sz + tr[tr[x].s[1]].sz + 1;
}
void pushdown(int x) {
    if (tr[x].flag) {
        swap(tr[x].s[0], tr[x].s[1]);
        tr[tr[x].s[0]].flag ^= 1;
        tr[tr[x].s[1]].flag ^= 1;
        tr[x].flag = 0;
    }
}
void rotate(int x) {
    int y = tr[x].p, z = tr[y].p;
    int k = tr[y].s[1] == x;
    tr[z].s[tr[z].s[1] == y] = x, tr[x].p = z;
    tr[y].s[k] = tr[x].s[k ^ 1], tr[tr[x].s[k ^ 1]].p = y;
    tr[x].s[k ^ 1] = y, tr[y].p = x;
    pushup(y), pushup(x);
}
void splay(int x, int k) {
    while (tr[x].p != k) {
        int y = tr[x].p, z = tr[y].p;
        if (z != k) {
            if ((tr[y].s[1] == x) ^ (tr[z].s[1] == y)) rotate(x);
            else rotate(y);
        }
        rotate(x);
    }
    if (!k) root = x;
}
void insert(int v) {
    int u = root, p = 0;
    while (u) {
        p = u;
        u = tr[u].s[v > tr[u].v] == p; // 按值比较时 u = tr[u].s[v > tr[u].v];
    }
    u = ++idx;
    if (p) tr[p].s[v > tr[p].v] = u;
    tr[u].init(v, p);
    splay(u, 0);
}
int get_k(int k) {
    int u = root;
    while (1) {
        pushdown(u);
        if (tr[tr[u].s[0]].sz >= k) u = tr[u].s[0];
        else if (tr[tr[u].s[0]].sz + 1 == k) return u; // 返回值看题目要求
        else {
            k -= tr[tr[u].s[0]].sz + 1;
            u = tr[u].s[1];
        }
    }
    return -1;
}
void output(int u) {
    pushdown(u);
    if (tr[u].s[0]) output(tr[u].s[0]);
    if (tr[u].v >= 1 && tr[u].v <= n) cout << tr[u].v << ' ';
    if (tr[u].s[1]) output(tr[u].s[1]);
}
void solve() {
    cin >> n >> m;
    for (int i = 0; i <= n + 1; i++) insert(i); // 多插入了无穷小和无穷大
    while (m--) {
        int l, r;
        cin >> l >> r; // 由于存在无穷小，所以应该修改 l + 1 到 r + 1，应该使用 l 和 r + 2
        l = get_k(l), r = get_k(r + 2);
        splay(l, 0), splay(r, l);
        tr[tr[r].s[0]].flag ^= 1;
    }
    output(root);
}
```

维护数列

1.在当前数列的第 pos 个数字后插入 tot 个数字

2.从当前数列的第 pos 个数字开始删除 tot 个数字

3.将从当前数列的第 pos 个数字开始的 tot 个数字修改为 c

4.将从当前数列的第 pos 个数字开始的 tot 个数字翻转

5.计算从当前数列的第 pos 个数字开始的 tot 个数字的和

6.求出整个序列的最大子段和(至少包含一个数字)

```
int n, m;
struct Node {
    int s[2], v, p;
    int rev, same;
    int sz, sum, ms, ls, rs;
    void init(int _v, int _p) {
        s[0] = s[1] = 0, v = _v, p = _p;
        rev = same = 0;
        sz = 1, sum = ms = v;
        ls = rs = max(v, 0);
    }
} tr[N];
int root, nodes[N], tt;
int w[N];
void pushup(int x) {
    auto &u = tr[x], &l = tr[u.s[0]], &r = tr[u.s[1]];
    u.sz = l.sz + r.sz + 1;
    u.sum = l.sum + r.sum + u.v;
    u.ls = max(l.ls, l.sum + u.v + r.ls);
    u.rs = max(r.rs, r.sum + u.v + l.rs);
    u.ms = max({l.ms, r.ms, l.rs + u.v + r.ls});
}
void pushdown(int x) {
    auto &u = tr[x], &l = tr[u.s[0]], &r = tr[u.s[1]];
    if (u.same) {
        u.same = u.rev = 0;
        if (u.s[0]) l.same = 1, l.v = u.v, l.sum = l.v * l.sz;
        if (u.s[1]) r.same = 1, r.v = u.v, r.sum = r.v * r.sz;
        if (u.v > 0) {
            if (u.s[0]) l.ms = l.ls = l.rs = l.sum;
            if (u.s[1]) r.ms = r.ls = r.rs = r.sum;
        }
        else {
            if (u.s[0]) l.ms = l.v, l.ls = l.rs = 0;
            if (u.s[1]) r.ms = r.v, r.ls = r.rs = 0;
        }
    }
    else if (u.rev) {
        u.rev = 0, l.rev ^= 1, r.rev ^= 1;
        swap(l.ls, l.rs), swap(r.ls, r.rs);
        swap(l.s[0], l.s[1]), swap(r.s[0], r.s[1]);
    }
}
void rotate(int x) {
    int y = tr[x].p, z = tr[y].p;
    int k = tr[y].s[1] == x;
    tr[z].s[tr[z].s[1] == y] = x, tr[x].p = z;
    tr[y].s[k] = tr[x].s[k ^ 1], tr[tr[x].s[k ^ 1]].p = y;
    tr[x].s[k ^ 1] = y, tr[y].p = x;
    pushup(y), pushup(x);
}
void splay(int x, int k) {
    while (tr[x].p != k) {
        int y = tr[x].p, z = tr[y].p;
        if (z != k) {
            if ((tr[y].s[1] == x) ^ (tr[z].s[1] == y)) rotate(x);
            else rotate(y);
        }
        rotate(x);
    }
    if (!k) root = x;
}
int get_k(int k) {
    int u = root;
    while (1) {
        pushdown(u);
        if (tr[tr[u].s[0]].sz >= k) u = tr[u].s[0];
        else if (tr[tr[u].s[0]].sz + 1 == k) return u;
        else {
```

```
            k -= tr[tr[u].s[0]].sz + 1;
            u = tr[u].s[1];
        }
    }
    return -1;
}
int build(int l, int r, int p) { // 以 p 为父亲建树
    int mid = l + r >> 1;
    int u = nodes[tt--];
    tr[u].init(w[mid], p);
    if (l < mid) tr[u].s[0] = build(l, mid - 1, u);
    if (r > mid) tr[u].s[1] = build(mid + 1, r, u);
    pushup(u);
    return u;
}
void dfs(int u) {
    if (tr[u].s[0]) dfs(tr[u].s[0]);
    if (tr[u].s[1]) dfs(tr[u].s[1]);
    nodes[++tt] = u;
}
void solve() {
    for (int i = 1; i < N; i++) nodes[++tt] = i; // 垃圾回收
    cin >> n >> m;
    tr[0].ms = w[0] = w[n + 1] = -INF;
    for (int i = 1; i <= n; i++) cin >> w[i];
    root = build(0, n + 1, 0);
    string op;
    while (m--) {
        cin >> op;
        if (op == "INSERT") {
            int pos, tot; // 插入位置和个数
            cin >> pos >> tot;
            for (int i = 0; i < tot; i++) cin >> w[i];
            int l = get_k(pos + 1), r = get_k(pos + 2); // 由于有哨兵要加 1
            splay(l, 0), splay(r, l);
            int u = build(0, tot - 1, r);
            tr[r].s[0] = u;
            pushup(r), pushup(l);
        }
        else if (op == "DELETE") {
            int pos, tot; // 删除位置和个数
            cin >> pos >> tot;
            int l = get_k(pos), r = get_k(pos + tot + 1);
            splay(l, 0), splay(r, l);
            dfs(tr[r].s[0]); // 将 r 的左子树删掉
            tr[r].s[0] = 0;
            pushup(r), pushup(l);
        }
        else if (op == "MAKE-SAME") {
            int pos, tot, c;
            cin >> pos >> tot >> c;
            int l = get_k(pos), r = get_k(pos + tot + 1);
            splay(l, 0), splay(r, l);
            auto &son = tr[tr[r].s[0]];
            son.same = 1, son.v = c, son.sum = c * son.sz;
            if (c > 0) son.ms = son.ls = son.rs = son.sum;
            else son.ms = c, son.ls = son.rs = 0;
            pushup(r), pushup(l);
        }
        else if (op == "REVERSE") {
            int pos, tot;
            cin >> pos >> tot;
            int l = get_k(pos), r = get_k(pos + tot + 1);
            splay(l, 0), splay(r, l);
            auto &son = tr[tr[r].s[0]];
            son.rev ^= 1;
            swap(son.ls, son.rs);
            swap(son.s[0], son.s[1]);
            pushup(r), pushup(l);
        }
        else if (op == "GET-SUM") {
            int pos, tot;
            cin >> pos >> tot;
            int l = get_k(pos), r = get_k(pos + tot + 1);
            splay(l, 0), splay(r, l);
            cout << tr[tr[r].s[0]].sum << '\n';
        }
        else if (op == "MAX-SUM") cout << tr[root].ms << '\n';
    }
}
```

ST表

```
void solve() {
    int n;
    cin >> n;
    vector<int> lg(n + 1, 0);
    vector<vector<int>> st(n + 1, vector<int> (20, 0));
    for (int i = 1; i <= n; i++) {
        cin >> st[i][0];
        if (i >= 2) lg[i] = lg[i >> 1] + 1;
    }
    for (int j = 1; j <= lg[n]; j++) {
        for (int i = 1; i <= n - (1 << j) + 1; i++) {
            st[i][j] = max(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
        }
    }
    auto query = [&](int l, int r) -> int {
        int x = lg[r - l + 1];
        return max(st[l][x], st[r - (1 << x) + 1][x]);
    };
}
```

Treap

```
int n;
struct Node {
    int l, r;
    int key, val;
    int cnt, sz;
} tr[N];
int root, idx;
bool exist;
void pushup(int p) {
    tr[p].sz = tr[tr[p].l].sz + tr[tr[p].r].sz + tr[p].cnt;
}
int get_node(int key) {
    tr[++idx].key = key;
    tr[idx].val = rand();
    tr[idx].cnt = tr[idx].sz = 1;
    return idx;
}
void build() {
    get_node(-INF), get_node(INF);
    root = 1, tr[1].r = 2;
    pushup(root);
}
void zig(int &p) { // 右旋
    int q = tr[p].l;
    tr[p].l = tr[q].r;
    tr[q].r = p;
    p = q;
    pushup(tr[p].r), pushup(p);
}
void zag(int &p) { // 左旋
    int q = tr[p].r;
    tr[p].r = tr[q].l;
    tr[q].l = p;
    p = q;
    pushup(tr[p].l), pushup(p);
}
void insert(int &p, int key) {
    if (!p) p = get_node(key);
    else if (tr[p].key == key) tr[p].cnt++;
    else if (tr[p].key > key) {
        insert(tr[p].l, key);
        if (tr[tr[p].l].val > tr[p].val) zig(p);
    }
    else {
        insert(tr[p].r, key);
        if (tr[tr[p].r].val > tr[p].val) zag(p);
    }
    pushup(p);
}
void remove(int &p, int key) {
    if (!p) return;
    if (tr[p].key == key) {
        if (tr[p].cnt > 1) tr[p].cnt--;
        else if (tr[p].l || tr[p].r) {
            if (!tr[p].r || tr[tr[p].l].val > tr[tr[p].r].val) {
```

```
                zig(p);
                remove(tr[p].r, key);
            }
            else {
                zag(p);
                remove(tr[p].l, key);
            }
        }
        else p = 0;
    }
    else if (tr[p].key > key) remove(tr[p].l, key);
    else remove(tr[p].r, key);
    pushup(p);
}
int get_rank_by_key(int p, int key) {
    if (!p) return 0;
    if (tr[p].key == key) {
        exist = 1;
        return tr[tr[p].l].sz + 1;
    }
    else if (tr[p].key > key) return get_rank_by_key(tr[p].l, key);
    else return tr[tr[p].l].sz + tr[p].cnt + get_rank_by_key(tr[p].r, key);
}
int get_key_by_rank(int p, int rank) {
    if (!p) return INF;
    if (tr[tr[p].l].sz >= rank) return get_key_by_rank(tr[p].l, rank);
    else if (tr[tr[p].l].sz + tr[p].cnt >= rank) return tr[p].key;
    else return get_key_by_rank(tr[p].r, rank - tr[tr[p].l].sz - tr[p].cnt);
}
int get_prev(int p, int key) {
    if (!p) return -INF;
    if (tr[p].key >= key) return get_prev(tr[p].l, key);
    else return max(tr[p].key, get_prev(tr[p].r, key));
}
int get_nxt(int p, int key) {
    if (!p) return INF;
    if (tr[p].key <= key) return get_nxt(tr[p].r, key);
    else return min(tr[p].key, get_nxt(tr[p].l, key));
}
void solve() {
    build();
    cin >> n;
    while (n--) {
        exist = 0;
        int opt, x;
        cin >> opt >> x;
        if (opt == 1) insert(root, x);
        else if (opt == 2) remove(root, x);
        else if (opt == 3) {
            int ans = get_rank_by_key(root, x) - 1;
            if (exist) cout << ans << '\n';
            else cout << ans + 1 << '\n';
        }
        else if (opt == 4) cout << get_key_by_rank(root, x + 1) << '\n';
        else if (opt == 5) cout << get_prev(root, x) << '\n';
        else cout << get_nxt(root, x) << '\n';
    }
}
```

Trie

0 号点既是根节点，又是空节点

son[][] 存储树中每个节点的子节点

cnt[] 存储以每个节点结尾的单词数量

```
int son[N][26], cnt[N], idx;
// 插入一个字符串
void insert(string str) {
    int p = 0;
    int len = str.size();
    for (int i = 0; i < len; i++) {
        int u = str[i] - 'a';
        if (!son[p][u]) son[p][u] = ++idx;
        p = son[p][u];
    }
    cnt[p]++;
}
// 查询字符串出现的次数
```

```
int query(string str) {
    int p = 0;
    int len = str.size();
    for (int i = 0; i < len; i++) {
        int u = str[i] - 'a';
        if (!son[p][u]) return 0;
        p = son[p][u];
    }
    return cnt[p];
}
```

最大异或对

```
const int N = 1e5 + 10, M = 31 * N;
int n, a[N], son[M][2], idx;
void insert(int x) {
    int p = 0;
    for (int i = 30; i >= 0; i--) {
        int u = x >> i & 1;
        if (!son[p][u]) son[p][u] = ++idx;
        p = son[p][u];
    }
}
int query(int x) {
    int p = 0;
    int res = 0;
    for (int i = 30; i >= 0; i--) {
        int u = x >> i & 1;
        if (son[p][!u]) {
            p = son[p][!u];
            res = res * 2 + !u;
        }
        else {
            p = son[p][u];
            res = res * 2 + u;
        }
    }
    return res;
}
void solve() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        insert(a[i]);
    }
    LL ans = 0;
    for (int i = 1; i <= n; i++) {
        LL now = query(a[i]);
        ans = max(ans, a[i] ^ now);
    }
    cout << ans << '\n';
}
```

数学

本源勾股数

a是奇数, b是偶数，c是奇数 a * a + b * b = c * c

a = s * t, c = (s * s + t * t) / 2, b = (s * s - t * t) / 2, t < s, t 和 s 为奇数且互素

多项式

65537 998244353 1004535809 4179340454199820289

NTT

```
namespace NTT {
    int limit;
    vector<int> A, B, rev;
    inline int add(int x, int y) { return x += y, x >= mod ? x - mod : x; }
    inline int mul(int x, int y) { return 1ll * x * y % mod; }
    int qpow(int x, int y) {
        int res = 1;
        for (; y; y >>= 1, x = mul(x, x)) if (y & 1) res = mul(res, x);
        return res;
    }
    void init() {
```

```
        int ed = n * 2, bit = -1;
        for (limit = 1; limit <= ed; limit <<= 1) ++bit;
        A.resize(limit); B.resize(limit); rev.resize(limit);
        for (int i = 0; i < limit; ++i)    rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << bit);
    }
    void ntt(vector<int>& P, int op) {
        for (int i = 0; i < limit; ++i) {
            if (i < rev[i])swap(P[i], P[rev[i]]);
        }
        for (int mid = 1; mid < limit; mid <<= 1) {
            int euler = qpow(3, (mod - 1) / (mid << 1));
            if (op < 0)    euler = qpow(euler, mod - 2);
            for (int i = 0, pos = mid << 1; i < limit; i += pos) {
                int wk = 1;
                for (int j = 0; j < mid; ++j, wk = mul(wk, euler)) {
                    int x = P[i + j], y = mul(wk, P[i + j + mid]);
                    P[i + j] = add(x, y), P[i + j + mid] = add(x, mod - y);
                }
            }
        }
        if (op > 0)    return;
        int inv = qpow(limit, mod - 2);
        for (int i = 0; i < limit; ++i)    P[i] = mul(P[i], inv);
    }
    void work() {
        for (int i = 0; i <= n; ++i) A[i] = i & 1 ? mod - infact[i] : infact[i];
        for (int i = 0; i <= n; ++i) B[i] = mul(g[i], infact[i]);
        ntt(A, 1), ntt(B, 1);
        for (int i = 0; i < limit; ++i) A[i] = mul(A[i], B[i]);
        ntt(A, -1);
    }
} using namespace NTT;
```

```
namespace Polynomial {
    using Poly = std::vector<int>;
    constexpr int P(998244353), G(3), L(1 << 23); // L : max length of dft
    inline void inc(int &x, int y) { (x += y) >= P ? x -= P : 0; }
    inline int mod(int64_t x) { return x % P; }
    inline int fpow(int x, int k = P - 2) {
        int r = 1;
        for (; k; k >>= 1, x = 1LL * x * x % P)
            if (k & 1) r = 1LL * r * x % P;
        return r;
    }
    int w[L], _ = [] {
        w[L / 2] = 1;
        for (int i = L / 2 + 1, x = fpow(G, (P - 1) / L); i < L; i++) w[i] = 1LL * w[i - 1] * x % P;
        for (int i = L / 2 - 1; i >= 0; i--) w[i] = w[i << 1];
        return 0;
    }();
    void dft(int *a, int n) {  // 原地 NTT（前置 assert：n 必须是 2 的幂）
        assert((n & n - 1) == 0);
        for (int k = n >> 1; k; k >>= 1) {
            for (int i = 0; i < n; i += k << 1) {
                for (int j = 0; j < k; j++) {
                    int y = a[i + j + k];
                    a[i + j + k] = 1LL * (a[i + j] - y + P) * w[k + j] % P;
                    inc(a[i + j], y);
                }
            }
        }
    }
    void idft(int *a, int n) {  //原地逆 NTT；末尾乘 n⁻¹ 并做一次 反转
        assert((n & n - 1) == 0);
        for (int k = 1; k < n; k <<= 1) {
            for (int i = 0; i < n; i += k << 1) {
                for (int j = 0; j < k; j++) {
                    int x = a[i + j], y = 1LL * a[i + j + k] * w[k + j] % P;
                    a[i + j + k] = x - y < 0 ? x - y + P : x - y;
                    inc(a[i + j], y);
                }
            }
        }
        for (int i = 0, inv = P - (P - 1) / n; i < n; i++) a[i] = 1LL * a[i] * inv % P;
        std::reverse(a + 1, a + n);
    }
    // 做 “半宽 DFT” → “再卷一次点” 可把长度 n 的 DFT 拓成 2n，在半分治求指数的场景中常用（你后面的 exp2/exp 用到）
    void dftDoubling(int *a, int n) {
        std::copy_n(a, n, a + n);
        idft(a + n, n);
```

```
    for (int i = 0; i < n; i++) a[n + i] = 1LL * w[n + i] * a[n + i] % P;
    dft(a + n, n);
}
// 返回大于等于 len 的最小 2^k
int norm(int n) { return 1 << std::__lg(n * 2 - 1); }
void norm(Poly &a) {
    if (!a.empty())
        a.resize(norm(a.size()));
}
void dft(Poly &a) { dft(a.data(), a.size()); }
void idft(Poly &a) { idft(a.data(), a.size()); }
inline Poly &dotEq(Poly &a, Poly b) { // 点乘
    assert(a.size() == b.size());
    for (int i = 0; i < a.size(); i++)
        a[i] = 1LL * a[i] * b[i] % P;
    return a;
}
inline Poly dot(Poly a, Poly b) { return dotEq(a, b); }
Poly operator*(const Poly &a, const Poly &b) { // 快速卷积
    int len = a.size() + b.size() - 1;
    if (a.size() <= 16 || b.size() <= 16) {
        Poly c(len);
        for (size_t i = 0; i < a.size(); i++)
            for (size_t j = 0; j < b.size(); j++)
                c[i + j] = (c[i + j] + 1LL * a[i] * b[j]) % P;
        return c;
    }
    int n = norm(len);
    Poly foo = a;
    foo.resize(n);
    dft(foo);
    if (&a == &b) dotEq(foo, foo);
    else {
        Poly bar = b;
        bar.resize(n);
        dft(bar);
        dotEq(foo, bar);
    }
    idft(foo);
    foo.resize(len);
    return foo;
}
Poly &operator*=(Poly &a, int b) { for (auto &x : a) x = 1LL * x * b % P; return a; }
Poly operator*(Poly a, int b) { return a *= b; }
Poly operator*(int a, Poly b) { return b *= a; }
Poly &operator/=(Poly &a, int b) { return a *= fpow(b); }
Poly operator/(Poly a, int b) { return a /= b; }
Poly &operator+=(Poly &a, Poly b) {
    a.resize(std::max(a.size(), b.size()));
    for (int i = 0; i < b.size(); i++) inc(a[i], b[i]);
    return a;
}
Poly operator+(Poly a, Poly b) { return a += b; }
Poly &operator-=(Poly &a, Poly b) {
    a.resize(std::max(a.size(), b.size()));
    for (int i = 0; i < b.size(); i++) inc(a[i], P - b[i]);
    return a;
}
Poly operator-(Poly a, Poly b) { return a -= b; }
Poly operator-(Poly a) { for (auto &x : a) x ? x = P - x : 0; return a; }
Poly &operator>>=(Poly &a, int x) {
    if (x >= (int)a.size()) a.clear();
    else a.erase(a.begin(), a.begin() + x);
    return a;
}
Poly &operator<<=(Poly &a, int x) {
    a.insert(a.begin(), x, 0);
    return a;
}
Poly operator>>(Poly a, int x) { return a >>= x; }
Poly operator<<(Poly a, int x) { return a <<= x; }
Poly invRec(Poly a) {  // 多项式逆
    int n = a.size();
    assert((n & n - 1) == 0);
    if (n == 1) return {fpow(a[0])};
    int m = n >> 1;
    Poly b = invRec(Poly(a.begin(), a.begin() + m)), c = b;
    b.resize(n);
    dft(a), dft(b), dotEq(a, b), idft(a);
    for (int i = 0; i < m; i++) a[i] = 0;
    for (int i = m; i < n; i++) a[i] = P - a[i];
```

```
    dft(a), dotEq(a, b), idft(a);
    for (int i = 0; i < m; i++) a[i] = c[i];
    return a;
}
Poly inverse(Poly a) {
    int n = a.size();
    norm(a);
    a = invRec(a);
    a.resize(n);
    return a;
}
Poly operator/(Poly a, Poly b) { // return: c(len = n - m + 1), a = b * c + r
    int n = a.size(), m = b.size();
    if (n < m) return {0};
    int k = 1 << std::__lg(n - m << 1 | 1);
    std::reverse(a.begin(), a.end());
    std::reverse(b.begin(), b.end());
    a.resize(k), b.resize(k), b = invRec(b);
    a = a * b;
    a.resize(n - m + 1);
    std::reverse(a.begin(), a.end());
    return a;
}
std::pair<Poly, Poly> div(Poly a, Poly b) { // c, r
    int m = b.size();
    Poly c = a / b;
    b = b * c;
    a.resize(m - 1);
    for (int i = 0; i < m - 1; i++) inc(a[i], P - b[i]);
    return {c, a};
}
Poly operator%(Poly a, Poly b) { // return: r(len = m - 1)
    return div(a, b).second;
}
Poly sqrt(Poly a) {
    int raw = a.size();
    int d = 0;
    while (d < raw && !a[d]) d++;
    if (d == raw) return a;
    assert(~d & 1); // if (d & 1) return {};
    norm(a >>= d);
    int len = a.size();
    Poly b(len), binv(1), bsqr{a[0]}, foo, bar; // sqrt, sqrt_inv, sqrt_sqr
    b[0] = [](int x) {
        int ans = 0;
        for (int i = 0, w = P - 1, o = G, p = 1; i < 25; i++) {
            int v = i > 22 ? i == 23 ? 7 : 17 : 2;
            for (w /= v; fpow(x, w) != 1; x = 1LL * x * o % P) {
                (ans += p) >= P - 1 ? ans -= P - 1 : 0;
            }
            p = 1LL * p * v % P;
            o = fpow(o, v);
        }
        x = fpow(G, (P - 1 - ans) >> 1);
        return x < P - x ? x : P - x;
    }(a[0]);
    binv[0] = fpow(b[0]); // assert(a[0] == 1), b[0] = binv[0] = 1;
    auto shift = [](int x) { return (x & 1 ? x + P : x) >> 1; }; // quick div 2
    for (int m = 1, n = 2; n <= len; m <<= 1, n <<= 1) {
        foo.resize(n), bar = binv;
        for (int i = 0; i < m; i++) {
            foo[i] = 0;
            int &x = foo[i + m] = a[i] + a[i + m] - bsqr[i];
            if (x >= P) x -= P;
            if (x < 0) x += P;
        }
        binv.resize(n);
        dft(foo), dft(binv), dotEq(foo, binv), idft(foo);
        for (int i = m; i < n; i++) b[i] = shift(foo[i]);
        // inv
        if (n == len) break;
        for (int i = 0; i < n; i++) foo[i] = b[i];
        bar.resize(n), binv = bar;
        dft(foo), dft(bar), bsqr = dot(foo, foo), idft(bsqr);
        dotEq(foo, bar), idft(foo);
        for (int i = 0; i < m; i++) foo[i] = 0;
        for (int i = m; i < n; i++) foo[i] = P - foo[i];
        dft(foo), dotEq(foo, bar), idft(foo);
        for (int i = m; i < n; i++) binv[i] = foo[i];
    }
    b <<= d / 2;
```

```
    b.resize(raw);
    return b;
}
Poly deriv(Poly a) { // 求导
    for (int i = 0; i + 1 < a.size(); i++) a[i] = (i + 1LL) * a[i + 1] % P;
    a.pop_back();
    return a;
}
std::vector<int> inv = {1, 1};
void updateInv(int n) {
    if ((int)inv.size() <= n) {
        int p = inv.size();
        inv.resize(n + 1);
        for (int i = p; i <= n; i++) inv[i] = 1LL * (P - P / i) * inv[P % i] % P;
    }
}
Poly integ(Poly a, int c = 0) {
    int n = a.size();
    updateInv(n);
    Poly b(n + 1);
    b[0] = c;
    for (int i = 0; i < n; i++) b[i + 1] = 1LL * inv[i + 1] * a[i] % P;
    return b;
}
Poly ln(Poly a) {
    int n = a.size();
    assert(a[0] == 1);
    a = inverse(a) * deriv(a);
    a.resize(n - 1);
    return integ(a);
}
// $O(n \log n)$, slower than exp2
Poly expNewton(Poly a) {
    int n = a.size();
    assert((n & n - 1) == 0);
    assert(a[0] == 0);
    if (n == 1) return {1};
    int m = n >> 1;
    Poly b = expNewton(Poly(a.begin(), a.begin() + m)), c;
    b.resize(n), c = ln(b);
    a.resize(n << 1), b.resize(n << 1), c.resize(n << 1);
    dft(a), dft(b), dft(c);
    for (int i = 0; i < n << 1; i++) a[i] = (1LL + P + a[i] - c[i]) * b[i] % P;
    idft(a);
    a.resize(n);
    return a;
}
// semi-relaxed-conv
// $O(n\log^2n)$
// $b = e^a, b' = a'b$
// $(n+1)b_{n+1} = \sum_{i=0}^n a'_ib_{n-i}$
// $nb_n = \sum_{i=0}^{n-1} a'_ib_{n - 1 - i}$
Poly exp2(Poly a) {
    if (a.empty()) return {};
    assert(a[0] == 0);
    int n = a.size();
    updateInv(n);
    for (int i = 0; i + 1 < n; i++) {
        a[i] = a[i + 1] * (i + 1LL) % P;
    }
    a.pop_back();
    Poly b(n);
    b[0] = 1;
    for (int m = 1; m < n; m++) {
        int k = m & -m, l = m - k, r = std::min(m + k, n);
        Poly p(a.begin(), a.begin() + (r - l - 1));
        Poly q(b.begin() + l, b.begin() + m);
        p.resize(k * 2), q.resize(k * 2);
        dft(p), dft(q);
        dotEq(p, q);
        idft(p);
        for (int i = m; i < r; i++) inc(b[i], p[i - l - 1]);
        b[m] = 1LL * b[m] * inv[m] % P;
    }
    return b;
}
// semi-relaxed-conv
// $O(\frac{n\log^2n}{\log\log n})$
Poly exp(Poly a) {
    if (a.empty()) return {};
    assert(a[0] == 0);
```

```
        int n = a.size();
        updateInv(n);
        for (int i = 0; i + 1 < n; i++) {
            a[i] = a[i + 1] * (i + 1LL) % P;
        }
        a.pop_back();
        Poly b(n);
        b[0] = 1;
        std::vector<Poly> val_a[6], val_b(n);
        for (int m = 1; m < n; m++) {
            int k = 1, d = 0;
            while (!(m / k & 0xf)) k *= 16, d++;
            int l = m & ~(0xf * k), r = std::min(n, m + k);
            if (k == 1) {
                for (int i = m; i < r; i++) {
                    for (int j = l; j < m; j++) {
                        b[i] = (b[i] + 1LL * b[j] * a[i - j - 1]) % P;
                    }
                }
            }
            else {
                assert(d < 6);
                if (val_a[d].empty()) val_a[d].resize(n);
                val_b[m] = Poly(b.begin() + (m - k), b.begin() + m);
                val_b[m].resize(k * 2);
                dft(val_b[m]);
                Poly res(k * 2);
                for (; l < m; l += k) {
                    auto &p = val_a[d][m - l - k];
                    if (p.empty()) {
                        p = Poly(a.begin() + (m - l - k), a.begin() + (r - l - 1));
                        p.resize(2 * k);
                        dft(p);
                    }
                    auto &q = val_b[l + k];
                    for (int i = 0; i < k * 2; i++) res[i] = (res[i] + 1LL * p[i] * q[i]) % P;
                }
                idft(res);
                for (int i = m; i < r; i++) inc(b[i], res[i - m + k - 1]);
            }
            b[m] = 1LL * b[m] * inv[m] % P;
        }
        return b;
    }
    Poly power(Poly a, int k) {
        int n = a.size();
        long long d = 0;
        while (d < n && !a[d]) d++;
        if (d == n) return a;
        a >>= d;
        int b = fpow(a[0]);
        norm(a *= b);
        a = exp(ln(a) * k) * fpow(b, P - 1 - k % (P - 1));
        a.resize(n);
        d *= k;
        for (int i = n - 1; i >= d; i--) a[i] = a[i - d];
        d = std::min(d, 1LL * n);
        for (int i = d; i; a[--i] = 0) ;
        return a;
    }
    Poly power(Poly a, int k1, int k2) { // k1 = k % (P - 1), k2 = k % P
        int n = a.size();
        long long d = 0;
        while (d < n && !a[d]) d++;
        if (d == n) return a;
        a >>= d;
        int b = fpow(a[0]);
        norm(a *= b);
        a = exp(ln(a) * k2) * fpow(b, P - 1 - k1 % (P - 1));
        a.resize(n);
        d *= k1;
        for (int i = n - 1; i >= d; i--) a[i] = a[i - d];
        d = std::min(d, 1LL * n);
        for (int i = d; i; a[--i] = 0) ;
        return a;
    }
}
using namespace Polynomial;
```

分治NTT

```
auto calc = [&](auto &&calc, int l, int r) -> Poly {
    if (l == r) return p[l];
    int mid = l + r >> 1;
    return calc(calc, l, mid) * calc(calc, mid + 1, r);
};
```

高精度

高精度加

高精度加,两个 vector 中都是倒着存的，且都为正数，输出需要倒序输出

```
vector<int> add(vector<int> &A, vector<int> &B) {
    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size()|| i < B.size(); i++) {
        if (i < A.size()) t += A[i];
        if (i < B.size()) t += B[i];
        C.push_back(t % 10);
        t /= 10;
    }
    if (t) C.push_back(t);
    return C;
}
```

高精度减法

A >= B 且都大于等于 0，倒序存储，倒序输出

```
vector<int> sub(vector<int> &A, vector<int> &B) {
    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size(); i++) {
        t = A[i] - t;
        if (i < B.size()) t -= B[i];
        C.push_back((t + 10) % 10);
        if (t < 0) t = 1;
        else t = 0;
    }
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
```

高精度乘以低精度

倒序

```
vector<int> mul(vector<int> &A, int b) {
    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size() || t; i++) {
        if (i < A.size()) t += A[i] * b;
        C.push_back(t % 10);
        t /= 10;
    }
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
```

高精度乘以高精度

倒序 不过最好使用 fft 来解决

```
vector<int> mul(vector<int> &A, vector<int> &B) {
    vector<int> C(A.size() + B.size());
    for (int i = 0; i < A.size(); i++) {
        for (int j = 0; j < B.size(); j++) {
            C[i + j] += A[i] * B[j];
        }
    }
    for (int i = 0, t = 0; i < C.size() || t; i++) {
        t += C[i];
        if (i >= C.size()) C.push_back(t % 10);
        else C[i] = t % 10;
        t /= 10;
    }
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
```

```
}
```

高精度除法

A 是正序的 C 最终返回值是倒序的，商 C 余 r

```
vector<int> div(vector<int> &A, int b, int &r) {
    vector<int> C;
    r = 0;
    for (int i = 0; i < A.size(); i++) {
        r = r * 10 + A[i];
        C.push_back(r / b);
        r %= b;
    }
    reverse(C.begin(),C.end());
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
```

大整数类

```
struct bign {
    int d[maxn], len;
    void clean() { while(len > 1 && !d[len - 1]) len--; }
    bign()          { memset(d, 0, sizeof(d)); len = 1; }
    bign(int num)   { *this = num; }
    bign(char* num) { *this = num; }
    bign operator = (const char* num) {
        memset(d, 0, sizeof(d)); len = strlen(num);
        for (int i = 0; i < len; i++) d[i] = num[len - 1 - i] - '0';
        clean();
        return *this;
    }
    bign operator = (int num) {
        char s[20]; sprintf(s, "%d", num);
        *this = s;
        return *this;
    }
    bign operator + (const bign& b) {
        bign c = *this; int i;
        for (i = 0; i < b.len; i++) {
            c.d[i] += b.d[i];
            if (c.d[i] > 9) c.d[i] %= 10, c.d[i + 1]++;
        }
        while (c.d[i] > 9) c.d[i++] %= 10, c.d[i]++;
        c.len = max(len, b.len);
        if (c.d[i] && c.len <= i) c.len = i + 1;
        return c;
    }
    bign operator - (const bign& b) {
        bign c = *this; int i;
        for (i = 0; i < b.len; i++) {
            c.d[i] -= b.d[i];
            if (c.d[i] < 0) c.d[i] += 10, c.d[i + 1]--;
        }
        while (c.d[i] < 0) c.d[i++] += 10, c.d[i]--;
        c.clean();
        return c;
    }
    bign operator * (const bign& b) const {
        int i, j; bign c; c.len = len + b.len;
        for (j = 0; j < b.len; j++) for (i = 0; i < len; i++)
            c.d[i + j] += d[i] * b.d[j];
        for (i = 0; i < c.len - 1; i++)
            c.d[i + 1] += c.d[i] / 10, c.d[i] %= 10;
        c.clean();
        return c;
    }
    bign operator / (const bign& b) {
        int i, j;
        bign c = *this, a = 0;
        for (i = len - 1; i >= 0; i--) {
            a = a * 10 + d[i];
            for (j = 0; j < 10; j++) if (a < b * (j + 1)) break;
            c.d[i] = j;
            a = a - b * j;
        }
        c.clean();
        return c;
    }
```

```
    bign operator % (const bign& b) {
        int i, j;
        bign a = 0;
        for (i = len - 1; i >= 0; i--) {
            a = a * 10 + d[i];
            for (j = 0; j < 10; j++) if (a < b * (j + 1)) break;
            a = a - b * j;
        }
        return a;
    }
    bign operator += (const bign& b) {
        *this = *this + b;
        return *this;
    }
    bool operator < (const bign& b) const {
        if (len != b.len) return len < b.len;
        for (int i = len - 1; i >= 0; i--)
            if (d[i] != b.d[i]) return d[i] < b.d[i];
        return false;
    }
    bool operator > (const bign& b) const {return b < *this;}
    bool operator <= (const bign& b) const {return !(b < *this);}
    bool operator >= (const bign& b) const {return !(*this < b);}
    bool operator != (const bign& b) const {return b < *this || *this < b;}
    bool operator == (const bign& b) const {return !(b < *this) && !(b > *this);}
    string str() const {
        char s[maxn] = {};
        for (int i = 0; i < len; i++) s[len - 1 - i] = d[i] + '0';
        return s;
    }
};
istream& operator >> (istream& in, bign& x) {
    string s;
    in >> s;
    x = s.c_str();
    return in;
}
ostream& operator << (ostream& out, const bign& x) {
    out << x.str();
    return out;
}
```

高斯消元

高斯消元解方程组

```
int n;
long double a[N][N];
int gauss() {
    int c, r;
    for (c = 0, r = 0; c < n; c++) {
        int t = r;
        for (int i = r; i < n; i++) {   // 找到绝对值最大的行
            if (fabs(a[i][c]) > fabs(a[t][c])) {
                t = i;
            }
        }
        if (fabs(a[t][c]) < 1e-8) continue;
        for (int i = c; i <= n; i++) {  // 将绝对值最大的行换到最顶端
            swap(a[r][i], a[t][i]);
        }
        for (int i = n; i >= c; i--) { // 将当前行的首位变成 1
            a[r][i] /= a[r][c];
        }
        for (int i = r + 1; i < n; i++) {   // 用当前行将下面所有的列消成 0
            if (fabs(a[i][c]) > 1e-8) {
                for (int j = n; j >= c; j--) {
                    a[i][j] -= a[i][c] * a[r][j];
                }
            }
        }
        r++;
    }
    if (r < n) {
        for (int i = r; i < n; i++) {
            if (fabs(a[i][n]) > 1e-8) return 2; // 无解
        }
        return 1;  // 有无穷多组解
    }
    for (int i = n - 1; i >= 0; i--) {   // 消成对角矩阵
```

```
        for (int j = i + 1; j <= n; j++) {
            a[i][n] -= a[i][j] * a[j][n];
        }
    }
    return 0; // 有唯一解
}
void solve () {
    cin >> n;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= n; j++) {
            cin >> a[i][j];
        }
    }
    int t = gauss();
    if (t == 0) {
        for (int i = 0; i < n; i++){
            cout << fixed << setprecision(4) << a[i][n] << '\n';
        }
    }
    else if (t == 1) cout << "Infinite group solutions" << '\n';
    else cout << "No solution";
}
```

高斯消元解异或方程组

```
int var;
int gauss() {
    int c, r;
    for (c = 0, r = 0; c < n; c++) {
        int t = r;
        for (int i = r; i < n; i++) {   // 找主元
            if (a[i][c]) {
                t = i;
                break;
            }
        }
        if (!a[t][c]) continue;
        for (int i = c; i <= n; i++) {  // 换行
            swap(a[r][i], a[t][i]);
        }
        for (int i = r + 1; i < n; i++) {   // 用当前行将下面所有的列消
            if (a[i][c]) {
                for (int j = c; j <= n; j++) {
                    a[i][j] ^= a[r][j];
                }
            }
        }
        r++;
    }
    var = n - r; // 共有多少个可变元，总的方案为 pow(2,var)
    if (r < n) {
        for (int i = r; i < n; i++) {
            if (a[i][n]) return 2; //无解
        }
        return 1; // 多组解
    }
    for (int i = n - 1; i >= 0; i--) {
        for (int j = i + 1; j <= n; j++) {
            a[i][n] ^= a[i][j] & a[j][n];
        }
    }
    return 0; // 唯一解
}
```

积性函数

f 是积性函数值，cnt[i]表示 i 这个数最小的质因数出现了几次

```
f[1] = 1;
for (int i = 2; i <= n; i++) {
    if (!not_prime[i]) {
        p[++tot] = i;
        f[i] = calc_f(i, 1);
    }
    for (int j = 1; j <= tot && p[j] <= n / i; j++) {
        not_prime[i * p[j]] = 1;
        if (i % p[j] == 0) {
            cnt[i * p[j]] = cnt[i] + 1;
            f[i * p[j]] = f[i] / calc_f(p[j], cnt[i]) * calc_f(p[j], cnt[i] + 1);
```

```
            break;
        }
        cnt[i * p[j]] = 1;
        f[i * p[j]] = f[i] * f[p[j]];
    }
}
```

极大线性无关组

两个数为 a 和 b（gcd == 1）最大不能表示出来的数为 (a - 1) * (b - 1) - 1,且不能表示数的个数为：(a - 1)(b - 1) / 2

完全背包方案数求极大线性无关组

```
int cnt;
int a[110], f[25010];
void solve() {
    int n;
    cin >> n;
    cnt = 0;
    memset(f, 0, sizeof(f));
    f[0] = 1;
    for (int i = 1; i <= n; i++) cin >> a[i];
    sort(a + 1, a + n + 1);
    int m = a[n];
    for (int i = 1; i <= n; i++) {
        for (int j = a[i]; j <= m; j++) {
            f[j] += f[j - a[i]];
        }
    }
    for (int i = 1; i <= n; i++) {
        if (f[a[i]] == 1) {
            cnt++; // 如果这个数可以被除了自己以外的数表示那么他就不是极大线性无关组的一员
        }
    }
    cout << cnt << '\n';
}
```

矩阵求逆

```
#include<bits/stdc++.h>
#define re register
#define il inline
#define ll long long
using namespace std;
const int N = 405, mod = 1e9 + 7;
int n;
ll a[N][N << 1];
il ll qpow(ll x, ll k) {
  ll ans = 1;
  while (k) {
    if (k & 1) ans = ans * x % mod;
    x = x * x % mod;
    k >>= 1;
  }
  return ans % mod;
}
il void Gauss_j() {
  for (re int i = 1, r; i <= n; ++i) {
    r = i;
    for (re int j = i + 1; j <= n; ++j)
      if (a[j][i] > a[r][i]) r = j;
    if (r != i) swap(a[i], a[r]);
    if (!a[i][i]) {
      puts("No Solution");
      return;
    }
    int kk = qpow(a[i][i], mod - 2);  // 求逆元
    for (re int k = 1; k <= n; ++k) {
      if (k == i) continue;
      int p = a[k][i] * kk % mod;
      for (re int j = i; j <= (n << 1); ++j)
        a[k][j] = ((a[k][j] - p * a[i][j]) % mod + mod) % mod;
    }
    for (re int j = 1; j <= (n << 1); ++j) a[i][j] = (a[i][j] * kk % mod);
    // 更新当前行 如果放在最后要再求一次逆元，不如直接放在这里
  }
  for (re int i = 1; i <= n; ++i) {
    for (re int j = n + 1; j < (n << 1); ++j) cout << a[i][j] << ' ';
    cout << a[i][n << 1] << '\n';
```

```
  }
}
int main() {
  cin >> n;
  for (re int i = 1; i <= n; ++i)
    for (re int j = 1; j <= n; ++j)
      cin >> a[i][j], a[i][i + n] = 1;
  Gauss_j();
    return 0;
}
```

快速幂

快速幂

```
LL qmi(i64 a, i64 k, i64 p) {
   i64 res = 1 % p;
   LL t = a;
   while (k) {
       if (k & 1) res = res * t % p;
       k >>= 1;
       t = t * t % p;
   }
   return res;
}
```

广义矩阵

```
const int INF = INT_MIN / 2;
struct Matrix {
    int a[2][2];
    Matrix() {
        for (int i = 0; i < 2; ++i)
            for (int j = 0; j < 2; ++j)
                a[i][j] = (i == j ? 0 : INF); // 单位矩阵
    }

    // Max-Plus 乘法：C[i][j] = max_k (A[i][k] + B[k][j])
    Matrix operator*(const Matrix &b) const {
        Matrix c;
        for (int i = 0; i < 2; ++i)
            for (int j = 0; j < 2; ++j) {
                c.a[i][j] = INF;
                for (int k = 0; k < 2; ++k) {
                    if (a[i][k] != INF && b.a[k][j] != INF)
                        c.a[i][j] = max(c.a[i][j], a[i][k] + b.a[k][j]);
                }
            }
        return c;
    }
};
```

矩阵快速幂

```
struct matrix {
    i64 x[2][2];
    matrix() { memset(x, 0, sizeof(x)); }
    matrix (i64 a[2][2]) {
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                x[i][j] = a[i][j];
            }
        }
    }
};
matrix add(matrix a, matrix b) {
    matrix res;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            res.x[i][j] = (a.x[i][j] + b.x[i][j]) % mod;
        }
    }
    return res;
}
matrix multiply(matrix a, matrix b) {
    matrix res;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
```

```
                res.x[i][j] = (res.x[i][j] + (a.x[i][k] * b.x[k][j]) % mod) % mod;
            }
        }
    }
    return res;
}
matrix mpow(matrix &a, i64 m) { // 矩阵 a 的 m 次方
    matrix res;
    for (int i = 0; i < 3; i++) res.x[i][i] = 1; // 单位矩阵
    while (m > 0) {
        if (m & 1) res = multiply(res, a);
        a = multiply(a, a);
        m >>= 1;
    }
    return res;
}
// 输出时也要输出((res.x[1][1] % mod) + mod) % mod
```

龟速乘

```
LL qadd(LL a, LL b, LL p) {
    LL res = 0;
    while (b) {
        if (b & 1) res = (res + a) % p;
        a = (a + a) % p;
        b >>= 1;
    }
    return res;
}
```

拉格朗日插值

求多项式系数 O(n^2)

```
#include<bits/stdc++.h>
using namespace std;
using i64 = long long;
constexpr int MOD = 998244353;
int inv(int k) {
    int res = 1;
    for (int e = MOD - 2; e; e /= 2) {
        if (e & 1) res = (i64)res * k % MOD;
        k = (i64)k * k % MOD;
    }
    return res;
}
// 返回 f 满足 f(x_i) = y_i
// 不考虑乘法逆元的时间，显然为 O(n^2)
vector<int> lagrange_interpolation(const vector<int> &x, const vector<int> &y) {
    const int n = x.size();
    vector<int> M(n + 1), xx(n), f(n);
    M[0] = 1;
    // 求出 M(x) = prod_(i=0..n-1)(x - x_i)
    for (int i = 0; i < n; i++) {
        for (int j = i; j >= 0; j--) {
            M[j + 1] = (M[j] + M[j + 1]) % MOD;
            M[j] = (i64)M[j] * (MOD - x[i]) % MOD;
        }
    }
    // 求出 xx_i = M'(x_i) = (M / (x - x_i)) mod (x - x_i) 一定非零
    for (int i = n - 1; i >= 0; i--) {
        for (int j = 0; j < n; j++) {
            xx[j] = ((i64)xx[j] * x[j] + (i64)M[i + 1] * (i + 1)) % MOD;
        }
    }
    // 组合出 f(x) = sum_(i=0..n-1)(y_i / M'(x_i))(M / (x - x_i))
    for (int i = 0; i < n; i++) {
        i64 t = (i64)y[i] * inv(xx[i]) % MOD, k = M[n];
        for (int j = n - 1; j >= 0; j--) {
            f[j] = (f[j] + k * t) % MOD;
            k = (M[j] + k * x[i]) % MOD;
        }
    }
    return f;
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, k;
```

```
    cin >> n >> k;
    vector<int> x(n), y(n);
    for (int i = 0; i < n; i++) cin >> x[i] >> y[i];
    const auto f = lagrange_interpolation(x, y);
    int v = 0;
    for (int i = n - 1; i >= 0; --i) v = ((i64)v * k + f[i]) % MOD;
    cout << v << '\n';
    return 0;
}
```

横坐标从 0 连续

求值只需 O(n)，但无法求多项式系数

```
const int N = 1e6 + 10, mod = 1e9 + 7;
i64 qmi(i64 a, i64 k) {
    i64 res = 1;
    while (k) {
        if (k & 1) res = res * a % mod;
        a = a * a % mod;
        k >>= 1;
    }
    return res;
}
int fact[N], infact[N];
void init() {
    fact[0] = infact[0] = 1;
    for (int i = 1; i < N; i++) fact[i] = (i64)fact[i - 1] * i % mod;
    infact[N - 1] = qmi(fact[N - 1], mod - 2);
    for (int i = N - 2; i >= 1; i--) infact[i] = (i64)infact[i + 1] * (i + 1) % mod;
}
// 结点固定为 x_i = i (0..n-1)，只求 f(k)
int lagrange_interpolation(const vector<int> &y, int k) {
    int n = (int)y.size();
    if (k < n) return y[k];
    // w_i = (-1)^{n-1-i} * y_i / (i! * (n-1-i)!)
    vector<int> w(n);
    for (int i = 0; i < n; i++) {
        i64 t = (i64)y[i] * infact[i] % mod * infact[n - 1 - i] % mod;
        if ((n - 1 - i) & 1) t = -t % mod;
        w[i] = t;
    }

    vector<i64> dif(n);
    for (int i = 0; i < n; i++) dif[i] = (k - i) % mod;

    // 前缀/后缀积，拿到：
    //   prodAll = ∏(k - i) = M(k)
    //   inv_dif[i] = 1 / (k - i)
    vector<int> pref(n + 1, 1), suf(n + 1, 1);
    for (int i = 0; i < n; i++) pref[i + 1] = (i64)pref[i] * dif[i] % mod;
    for (int i = n - 1; i >= 0; i--) suf[i] = (i64)suf[i + 1] * dif[i] % mod;
    int prodAll = pref[n];                // M(k)
    int invProd = qmi(prodAll, mod - 2);           // 一次求逆
    // inv_dif[i] = pref[i] * suf[i+1] / prodAll
    i64 sum = 0;
    for (int i = 0; i < n; i++) {
        int inv_dif_i = (i64)pref[i] * suf[i + 1] % mod * invProd % mod;
        sum = (sum + (i64)w[i] * inv_dif_i % mod) % mod;
    }
    i64 res = (prodAll * sum % mod + mod) % mod;
    return res;
}
```

曼哈顿，切比雪夫

将一个点 (x, y) 的坐标变为 (x + y, x − y) 后, 原坐标系中的曼哈顿距离 = 新坐标系中的切比雪夫距离

将一个点 (x, y) 的坐标变为 ((x + y) / 2, (x − y) /2) 后, 原坐标系中的切比雪夫距离 = 新坐标系中的曼哈顿距离

莫比乌斯反演

容斥如果大的集合是小集合的子集，那么可以先求大的然后类似埃氏筛从小的集合里筛掉

已知（因数和）

$$f(n) = \sum_{d|n} g(d). \tag{2}$$

则

$$g(n) = \sum_{d|n} \mu(d)\, f\Big(\frac{n}{d}\Big) = \sum_{d|n} \mu\Big(\frac{n}{d}\Big)\, f(d). \tag{3}$$

等价的狄利克雷卷积写法：

$$f = g * \mathbf{1}, \qquad g = f * \mu, \tag{4}$$

其中 $\mathbf{1}(n) \equiv 1$，μ 为莫比乌斯函数。

已知（倍数和）：

$$f(n) = \sum_{\substack{d \geq 1 \\ n|d}} g(d) = \sum_{k \geq 1} g(kn). \tag{5}$$

则（莫比乌斯反演，倍数版）：

$$g(n) = \sum_{\substack{d \geq 1 \\ n|d}} \mu\left(\frac{d}{n}\right) f(d) = \sum_{k \geq 1} \mu(k)\, f(kn). \tag{6}$$

其中 μ 为莫比乌斯函数。

$$[\gcd(x, y) = 1] = \sum_{d \,|\, \gcd(x,y)} \mu(d) = \sum_{\substack{d \,|\, x \\ d \,|\, y}} \mu(d). \tag{7}$$

狄利克雷卷积：

$$(h = f * g)(x) := \sum_{d|x} f(d)\, g\Big(\frac{x}{d}\Big). \tag{8}$$

单位元：

$$\varepsilon(n) := [n = 1], \qquad \varepsilon(1) = 1,\ \varepsilon(n > 1) = 0. \tag{9}$$

基本恒等式：

$$\mu * \mathbf{1} = \varepsilon, \qquad \mathrm{id} = \varphi * \mathbf{1}, \qquad \varphi = \mu * \mathrm{id}. \tag{10}$$

展开式：

$$\sum_{d|n} \mu(d) = \varepsilon(n) = [n = 1], \tag{11}$$

$$\sum_{d|n} \varphi(d) = n = \mathrm{id}(n), \tag{12}$$

$$\varphi(n) = \sum_{d|n} \mu(d)\, \mathrm{id}\Big(\frac{n}{d}\Big) = \sum_{d|n} \mu(d)\, \frac{n}{d}. \tag{13}$$

线性筛求 mobius 函数

```
int primes[N], cnt;
bool st[N];
int mobius[N];
void init(int n) {
    mobius[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (st[i] == 0) {
            primes[cnt++] = i;
            mobius[i] = -1;
        }
        for (int j = 0; primes[j] * i <= n; j++) {
            int t = primes[j] * i;
            st[t] = 1;
            if (i % primes[j] == 0) {
                mobius[t] = 0;
                break;
            }
            mobius[t] = mobius[i] * -1;
        }
    }
}
```

整数分块

```
{
    int a, b, d;
    cin >> a >> b >> d;
    a /= d, b /= d;
    LL res = 0;
    int n = min(a, b);
    for (int l = 1, r; l <= n; l = r + 1) {
        r = min(n, min(a / (a / l), b / (b / l)));
        res += (sum[r] - sum[l - 1]) * (LL)(a / l) * (b / l);
    }
    cout << res << '\n';
}
```

逆元

线性求 1 到 n 的逆元

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

线性求任意 n 个数的逆元

```
s[0] = 1;
for (int i = 1; i <= n; i++) s[i] = s[i - 1] * a[i] % p; // 前缀积
sv[n] = qmi(s[n], p - 2);
// 当然这里也可以用 exgcd 来求逆元.
for (int i = n; i >= 1; --i) sv[i - 1] = sv[i] * a[i] % p; // 求每个前缀积的逆元
for (int i = 1; i <= n; ++i) inv[i] = sv[i] * s[i - 1] % p; // 求第 i 个数的逆元
```

线性求阶乘逆元

```
fact[0] = infact[0] = 1;
for (int i = 1; i < N; i++) fact[i] = (LL)fact[i - 1] * i % mod;
infact[N - 1] = qmi(fact[N - 1], mod - 2, mod);
for (int i = N - 2; i >= 1; i--) infact[i] = infact[i + 1] * (i + 1) % mod;
```

欧拉

求欧拉函数

```
int phi(int x) {
    int res = x;
    for (int i = 2; i <= x / i; i++)
        if (x % i == 0) {
            res = res / i * (i - 1);
            while (x % i == 0) x /= i;
        }
    if (x > 1) res = res / x * (x - 1);
    return res;
}
```

筛法求欧拉函数

```
int primes[N], cnt;     // primes[] 存储所有素数
int euler[N];           // 存储每个数的欧拉函数
bool st[N];         // st[x]存储 x 是否被筛掉
void get_eulers(int n) {
    euler[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (!st[i]) {
            primes[cnt++] = i;
            euler[i] = i - 1;
        }
        for (int j = 0; primes[j] * i <= n; j++) {
            int t = primes[j] * i;
            st[t] = true;
            if (i % primes[j] == 0) {
                euler[t] = euler[i] * primes[j];
                break;
            }
            euler[t] = euler[i] * (primes[j] - 1);
        }
    }
}
```

扩展欧拉定理

```
if (gcd(a, m) == 1) a ^ b % m = a ^ (b % phi(m)) % m;
else {
    if (b < phi(m)) a ^ b % m = a ^ b % m;
    else a ^ b % m = a ^ (b % phi(m) + phi(m)) % m;
}
```

生成函数

几何级数展开：

$$(1-x)^{-1}=\sum_{k=0}^{\infty}x^k=1+x+x^2+\cdots. \tag{14}$$

收敛条件：$|x|<1$（或视为形式幂级数则恒成立）。

几何级数展开：

$$(1-ax)^{-1}=\sum_{k=0}^{\infty}(ax)^k=\sum_{k=0}^{\infty}a^kx^k=1+ax+a^2x^2+\cdots. \tag{15}$$

收敛条件：$|ax|<1$（或作形式幂级数则恒成立）。

几何级数展开（$r\in\mathbb{Z}_{>0}$）：

$$(1-x^r)^{-1}=\sum_{k=0}^{\infty}(x^r)^k=\sum_{k=0}^{\infty}x^{kr}=1+x^r+x^{2r}+\cdots. \tag{16}$$

收敛条件：$|x^r|<1$（例如实数情形下 $|x|<1$；作为形式幂级数则恒成立）。

$$(1-x)^{-2}=\sum_{k=0}^{\infty}(k+1)\,x^k=1+2x+3x^2+\cdots. \tag{17}$$

（收敛条件：$|x|<1$；作为形式幂级数恒成立。）

广义二项式（负整数幂）：

$$(1-x)^{-n}=\sum_{k=0}^{\infty}\binom{n+k-1}{k}x^k=\sum_{k=0}^{\infty}\binom{n+k-1}{n-1}x^k,\qquad n\in\mathbb{Z}_{>0}. \tag{18}$$

（收敛条件：$|x|<1$；作为形式幂级数恒成立。）

广义二项式定理：

$$(1+x)^a=\sum_{k=0}^{\infty}c[a][k]\,x^k,\qquad c[a][k]=\binom{a}{k}=\frac{a(a-1)\cdots(a-k+1)}{k!},\ c[a][0]=1. \tag{19}$$

收敛条件：若 $a\notin\mathbb{Z}_{\geq 0}$，则级数在 $|x|<1$ 收敛；若 $a\in\mathbb{Z}_{\geq 0}$，则和式在 $k>a$ 处截断为有限多项式。

指数函数的幂级数展开：

$$\exp(x)=e^x=\sum_{n=0}^{\infty}\frac{x^n}{n!}=1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots. \tag{20}$$

（收敛半径为无穷大。）

自然对数的幂级数展开：

$$\ln(1+x)=\sum_{n=1}^{\infty}\frac{(-1)^{\,n+1}}{n}\,x^n=x-\frac{x^2}{2}+\frac{x^3}{3}-\cdots. \tag{21}$$

收敛性：$|x|<1$；当 $x=1$ 时收敛到 $\ln 2$，当 $x=-1$ 发散。

多重集合（用常生成函数 OGF）
从 `n` 类物品中取 `k` 个（允许重复）：

$$G(x)=\prod_{i=1}^{n}\left(1+x+x^2+\cdots\right)=(1-x)^{-n},\qquad \#(\text{size }k)=[x^k]\,G(x)=\binom{n+k-1}{k}. \tag{22}$$

若第 (i) 类至多取 (m_i) 个：

$$G(x)=\prod_{i=1}^{n}\left(1+x+\cdots+x^{m_i}\right),\qquad \#(\text{size }k)=[x^k]\,G(x). \tag{23}$$

排成一排（用指数生成函数 EGF）
将 (k) 个互异（带标签）物品排成一排（长度 (k) 的排列）：

$$P(z)=\sum_{k\geq 0}\frac{k!}{k!}z^k=\frac{1}{1-z},\qquad \#(k)=k!. \tag{24}$$

更一般地，若基本类 (A) 的 EGF 为 (A(z))，则

$$\mathrm{SEQ}_k(A):\quad \mathrm{EGF}=A(z)^k,\qquad \mathrm{SEQ}(A):\quad \mathrm{EGF}=\frac{1}{1-A(z)}, \tag{25}$$

且“大小为 (n)、长度为 (k)”的计数为

$$\#_{n,k}=n![z^n]\big(A(z)^k\big). \tag{26}$$

位运算

a ^ b == a + b - 2 (a & b)

1 - n 中第 k 位为 1 的个数为 ⌊n / (2 ^ (i + 1))⌋ * 2 ^ i + min(max(n mod (2 ^ (i + 1)) - (1ll << i) + 1, 0ll), 1ll << i)

全集 (1 << n) - 1 补集 ((1 << n) - 1) ^ s 差集 a & ~b 删除最小元素 s & (s - 1)

1 的个数 **builtin_popcount(s) 二进制长度** lg(s) + 1 集合最大元素 **lg(s) 集合最小元素** builtin_ctz(s)

s = 0 时**lg(0) 和** builtin_ctz(0) 是未定义行为 long long，需使用相应的 __builtin_popcountll

枚举非空子集

单次枚举是是 O(2^|s|), 枚举所有集合的子集是(3^n)

```
for (int sub = s; sub; sub = (sub - 1) & s) {
    // 处理 sub 的逻辑
}
```

枚举子集（包含空集）

```
int sub = s;
do {
    // 处理 sub 的逻辑
    sub = (sub - 1) & s;
} while (sub != s);
```

枚举全集 U 的所有大小恰好为 k 的子集

O(C[n][k] * n) k不为0

```
void GospersHack(int n, int k) {
    int cur = (1 << k) - 1;
    int limit = (1 << n);
    while (cur < limit) {
        // do something
        int lb = cur & -cur;
        int r = cur + lb;
        cur = ((r ^ cur) >> __builtin_ctz(lb) + 2) | r;
    }
}
```

枚举超集

```
for (int s = t; s < (1 << n); s = (s + 1) | t) {
    // 处理 s 的逻辑
}
```

线性基

异或线性基 最大值为所有线性基异或和，最小值为最小的线性基

如果r < n 线性相关 r == n 线性无关

不同子集异或和个数 1 << r

所有子集异或和之和 所有线性基异或和 * (1 << (r - 1))

最大子数组异或和 先求前缀异或，构造线段树，每个节点保存当前区间的所以前缀异或得线性基，查询区间线段树，合并两基后直接求最大异或。

```
int n;
i64 a[N];
int id[66];
bool canRepresent(LL x) {
    for (int i = 62; i >= 0; i--) {
        if ((x >> i & 1) && id[i] != -1) {
            x ^= a[id[i]];
```

```
        }
    }
    return x == 0;
}
void solve() {
    cin >> n;
    for (int i = 0; i < n; i++) cin >> a[i];
    memset(id, -1, sizeof(id));
    int r = 0;
    for (int i = 62; i >= 0; i--) {
        for (int j = r; j < n; j++) {
            if (a[j] >> i & 1) {
                swap(a[j], a[r]);
                break;
            }
        }
        if ((a[r] >> i & 1) == 0) continue;
        id[i] = r; // 记录每一位的主人
        for (int j = 0; j < n; j++) {
            if ((a[j] >> i & 1) && j != r) a[j] ^= a[r];
        }
        r++;
        if (r == n) break;
    }
    i64 res = 0;
    for (int i = 0; i < r; i++) res ^= a[i]; // 能表示的最大值
    cout << res << '\n';
    reverse(a, a + r);  // 查询第 k 小的结果(至少选一个进行异或)
    int q;
    cin >> q;
    while (q--) {
        i64 k;
        cin >> k;
        if (r < n) k--; // 能凑出 0
        if (k >= (1ll << r)) cout << -1 << '\n';
        else {
            i64 res = 0;
            for (int p = 0; p < r; p++) if (k >> p & 1) res ^= a[p];
            cout << res << '\n';
        }
    }
}
```

```
class XorBasis {
    static const int MAX_BIT = 61;
    long long b[MAX_BIT];
    int sz = 0;
    vector<long long> basis; // 缓存规约后的基向量
    bool rebuilt = false;     // 标记是否已规约
    // 内部规约函数
    void rebuild() {
        basis.clear();
        for (int i = 0; i < MAX_BIT; i++) {
            if (b[i]) basis.push_back(b[i]);
        }
        int n = basis.size();
        vector<int> pos(n); // 记录每个基向量的最高 1 位
        for (int i = 0; i < n; i++) {
            for (int j = MAX_BIT - 1; j >= 0; j--) {
                if (basis[i] >> j & 1) {
                    pos[i] = j;
                    break;
                }
            }
        }
        // 高斯消元，消除低位
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (i != j && (basis[i] >> pos[j] & 1)) {
                    basis[i] ^= basis[j];
                }
            }
        }
        // 按最高位排序
        vector<pair<int, long long>> tmp;
        for (int i = 0; i < n; i++) {
            for (int j = MAX_BIT - 1; j >= 0; j--) {
                if (basis[i] >> j & 1) {
                    tmp.emplace_back(j, basis[i]);
                    break;
```

```
                }
            }
        }
        sort(tmp.begin(), tmp.end());
        basis.clear();
        for (auto [_, val] : tmp) basis.push_back(val);
        rebuilt = true;
    }
public:
    XorBasis() { fill(b, b + MAX_BIT, 0); }
    // 插入一个数进线性基
    void insert(long long x) {
        for (int i = MAX_BIT - 1; i >= 0; i--) {
            if ((x >> i) & 1) {
                if (!b[i]) {
                    b[i] = x;
                    sz++;
                    rebuilt = false; // 标记需要重新规约
                    return;
                }
                x ^= b[i];
            }
        }
    }
    // 判断某个数能否由当前基表示
    bool canRepresent(long long x) const {
        for (int i = MAX_BIT - 1; i >= 0; i--) {
            if ((x >> i) & 1) {
                if (!b[i]) return false;
                x ^= b[i];
            }
        }
        return true;
    }
    // 返回最大异或和
    long long maxXor() const {
        long long res = 0;
        for (int i = MAX_BIT - 1; i >= 0; i--) {
            if ((res ^ b[i]) > res) {
                res ^= b[i];
            }
        }
        return res;
    }
    // 返回最小异或和
    long long minXor() const {
        if (!rebuilt) {
            const_cast<XorBasis*>(this)->rebuild();
        }
        return basis.empty() ? 0 : basis[0];
    }
    // 返回第 k 小异或值（0-based）
    long long kthXor(unsigned long long k) const {
        if (!rebuilt) {
            const_cast<XorBasis*>(this)->rebuild(); // 惰性规约
        }
        int n = basis.size();
        if (k >= (1ULL << n)) return -1; // 超出范围
        long long res = 0;
        for (int i = 0; i < n; i++) {
            if (k >> i & 1) {
                res ^= basis[i];
            }
        }
        return res;
    }
    // 获取线性基的大小（秩）
    int size() const {
        return sz;
    }
    // 获取规约后的线性基向量集合
    vector<long long> getBasis() const {
        if (!rebuilt) {
            const_cast<XorBasis*>(this)->rebuild(); // 惰性规约
        }
        return basis;
    }
};
```

行列式求值

O(n^3) 利用辗转相除法

```
int n, p;
int a[N][N];
int cal(int a[][N], int n, const int p) {
  int i, j, k, r = 1, fh = 0, l;
  for (i = 1; i <= n; i++) {
    k = i;
    for (j = i; j <= n; j++) {
            if (a[j][i]) {
                k = j;
                break;
            }
        }
    if (a[k][i] == 0) return 0;
    for (++j; j <= n; j++) if (a[j][i] && a[j][i] < a[k][i]) k = j;
    if (i != k) {
            swap(a[k], a[i]);
            fh ^= 1;
        }
    for (j = i + 1; j <= n; j++) {
      if (a[j][i] > a[i][i]) {
                swap(a[j], a[i]);
                fh ^= 1;
            }
      while (a[j][i]) {
        l = a[i][i] / a[j][i];
        for (k = i; k <= n; k++) a[i][k] = (a[i][k] + (LL)(p - l) * a[j][k]) % p;
        swap(a[j], a[i]);
                fh ^= 1;
      }
    }
    r = (LL)r * a[i][i] % p;
  }
  if (fh) return (p - r) % p;
  return r;
}
void solve() {
    cin >> n >> p;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            cin >> a[i][j];
            a[i][j] %= p;
        }
    }
    cout << cal(a, n, p);
}
```

原根

只有模 1, 2, 4, p ^ a, 2 * p ^ a 才存在原根 其中 p 是奇素数

m >= 3 (g, m) = 1, g 是 m 的原根当且仅当对于任意 phi(m) 的质因子 q, g ^ (phi(m) / q) 不同余 1

找到了最小原根 g 那么对于 (x, phi(m)) = 1的 x，有 g^x % m 也是 m 的原根

最小原根从1开始暴力枚举即可，枚举大小最坏不超过 m^0.25

找出所有原根

```
const int N = 1e6 + 10;
bool have_yg[N], vis[N];
int primes[N], cnt, phi[N];
void init() {
    phi[1] = 1;
    for (int i = 2; i < N; i++) {
        if (!vis[i]) {
            primes[cnt++] = i;
            phi[i] = i - 1;
        }
        for (int j = 0; i * primes[j] < N; j++) {
            vis[i * primes[j]] = 1;
            if (i % primes[j] == 0) {
                phi[i * primes[j]] = phi[i] * primes[j];
                break;
            }
            phi[i * primes[j]] = phi[i] * (primes[j] - 1);
        }
    }
```

```
    have_yg[2] = have_yg[4] = 1;
    for (int i = 1; i < cnt; i++) {
        i64 now = 1;
        while (now * primes[i] < N) {
            now *= primes[i];
            have_yg[now] = 1;
            if (now * 2 < N) have_yg[now * 2] = 1;
        }
    }
}
i64 qmi(i64 a, i64 k, i64 p) {
    i64 res = 1;
    while (k) {
        if (k & 1) res = res * a % p;
        a = a * a % p;
        k >>= 1;
    }
    return res;
}
void solve() {
    int n, d;
    cin >> n >> d;
    if (!have_yg[n]) {
        cout << 0 << '\n';
        cout << '\n';
        return;
    }
    if (n == 2) {
        cout << 1 << '\n';
        if (d == 1) cout << 1 << '\n';
        else cout << '\n';
        return;
    }
    vector<int> fact;
    int tmp = phi[n];
    for (int i = 2; i * i <= tmp; i++) {
        if (tmp % i == 0) {
            fact.push_back(i);
            while (tmp % i == 0) tmp /= i;
        }
    }
    if (tmp > 1) fact.push_back(tmp);
    auto check = [&](int x) -> bool {
        for (auto u : fact) {
            if (qmi(x, phi[n] / u, n) == 1)
                return 0;
        }
        return 1;
    };
    i64 aim;
    for (int i = 1; i <= n - 1; i++) {
        if (__gcd(i, n) != 1) continue;
        if (check(i)) {
            aim = i;
            break;
        }
    }
    vector<int> ans;
    for (int i = 1; i <= phi[n]; i++) {
        if (__gcd(i, phi[n]) == 1) {
            ans.push_back(qmi(aim, i, n));
        }
    }
    sort(ans.begin(), ans.end());
    cout << ans.size() << '\n';
    for (int i = d; i <= (int)ans.size(); i += d) cout << ans[i - 1] << ' ';
    cout << '\n';
}
```

质数和约数

已知

$$N = \prod_{i=1}^{k} p_i^{c_i} \qquad (p_i \text{ 为互异素数},\ c_i \in \mathbb{Z}_{\geq 0}). \tag{27}$$

约数个数（$\tau(N)$）：

$$\tau(N) = \prod_{i=1}^{k} (c_i + 1). \tag{28}$$

约数之和（$\sigma(N)$）：

$$\sigma(N) = \prod_{i=1}^{k} (p_i^0 + p_i^1 + \cdots + p_i^{c_i}) = \prod_{i=1}^{k} \frac{p_i^{c_i+1} - 1}{p_i - 1}. \tag{29}$$

分治求约数之和，a^b

```
LL qmi(LL a, LL k, int p) {
    LL res = 1;
    while (k) {
        if (k & 1) res = res * a % p;
        a = a * a % p;
        k >>= 1;
    }
    return res;
}
LL sum(LL val, LL num) {
    LL res = 1;
    if (num == 1) return res;
    if (num & 1) {
        res = (1 + val * sum(val, num - 1)) % mod;
        return res;
    }
    else {
        res = (1 + qmi(val, num / 2, mod)) * sum(val, num / 2) % mod;
        return res;
    }
}
void solve() {
    LL a, b;
    cin >> a >> b;
    LL res = 1;
    vector<pair<LL, LL>> tmp;
    LL op = a;
    for (int i = 2; i <= op / i; i++) {
        if (op % i == 0) {
            int cnt = 0;
            while (op % i == 0) {
                cnt++;
                op /= i;
            }
            tmp.push_back({(LL)i, cnt});
        }
    }
    if (op > 1) tmp.push_back({op, 1});
    if (a == 0) {
        cout << 0 << '\n';
        return;
    }
    for (auto [val, num] : tmp) res = (res * sum(val, num * b + 1)) % mod;
    cout << res << '\n';
}
```

gcd

```
int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}
int gcd(int m, int n) { // stein 算法求最大公约数
  if (m == n) return m;
  if (m < n) return gcd(n, m);
  if (m % 2 == 0) {
    if (n % 2 == 0) return 2 * gcd(m / 2, n / 2);
    else return gcd(m / 2, n);
  }
  else {
    if (n % 2 == 0) return gcd(m, n / 2);
    else return gcd(n, m - n);
  }
}
```

exgcd

求 x, y，使得 ax + by = gcd(a, b)

ax + by = c的 x 的通解为 x = x * (c / gcd) + k * (b / gcd), y = y * (c / gcd) - k * (a / gcd)

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

线性筛

```
int primes[N], cnt;     // primes[] 存储所有素数
bool vis[N];          // st[x] 存储 x 是否被筛掉
void get_primes(int n) {
    for (int i = 2; i <= n; i++) {
        if (!vis[i]) primes[cnt++] = i;
        for (int j = 0; primes[j] * i <= n; j++) {
            vis[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}
```

快速分解质因数 O(n^0.25)

```
i64 mul(i64 a, i64 b, i64 m) {
    return static_cast<__int128>(a) * b % m;
}
i64 power(i64 a, i64 b, i64 m) {
    i64 res = 1 % m;
    for (; b; b >>= 1, a = mul(a, a, m))
        if (b & 1)
            res = mul(res, a, m);
    return res;
}
bool isprime(i64 n) { // O(12 logn)
    if (n < 2) return false;
    static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
    int s = __builtin_ctzll(n - 1);
    i64 d = (n - 1) >> s;
    for (auto a : A) {
        if (a == n) return true;
        i64 x = power(a, d, n);
        if (x == 1 || x == n - 1) continue;
        bool ok = false;
        for (int i = 0; i < s - 1; i++) {
            x = mul(x, x, n);
            if (x == n - 1) {
                ok = true;
                break;
            }
        }
        if (!ok) return false;
    }
    return true;
}
vector<i64> factorize(i64 n) {
    vector<i64> p;
    function<void(i64)> f = [&](i64 n) {
        if (n <= 10000) {
            for (int i = 2; i * i <= n; i++)
                for (; n % i == 0; n /= i)
                    p.push_back(i);
            if (n > 1) p.push_back(n);
            return;
        }
        if (isprime(n)) {
            p.push_back(n);
            return;
        }
        auto g = [&](i64 x) {
            return (mul(x, x, n) + 1) % n;
        };
        i64 x0 = 2;
        while (true) {
            i64 x = x0;
            i64 y = x0;
            i64 d = 1;
```

```
            i64 power = 1, lam = 0;
            i64 v = 1;
            while (d == 1) {
                y = g(y);
                ++lam;
                v = mul(v, abs(x - y), n);
                if (lam % 127 == 0) {
                    d = __gcd(v, n);
                    v = 1;
                }
                if (power == lam) {
                    x = y;
                    power *= 2;
                    lam = 0;
                    d = __gcd(v, n);
                    v = 1;
                }
            }
            if (d != n) {
                f(d);
                f(n / d);
                return;
            }
            ++x0;
        }
    };
    f(n);
    sort(p.begin(), p.end());
    return p;
}
```

中国剩余定理

CRT

需保证模数互质

```
int n;
LL m[11], a[11];
LL exgcd(LL a, LL b, LL &x, LL &y) {
    if (!b) {
        x = 1; y = 0;
        return a;
    }
    LL d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
void solve() {
    cin >> n;
    LL M = 1;
    for (int i = 1; i <= n; i++) {
        cin >> m[i] >> a[i];
        M *= m[i];
    }
    LL res = 0;
    for (int i = 1; i <= n; i++) {
        LL Mi = M / m[i];
        LL inv, y;
        exgcd(Mi, m[i], inv, y);
        res = (res + (i128)a[i] * Mi * inv) % M;
    }
    cout << (res % M + M) % M;
}
```

EXCRT

模数可以不互质

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

```
    cin >> n;
    LL a1, m1;
    cin >> m1 >> a1;
    for (int i = 1; i <= n - 1; i++) {
        LL a2, m2;
        cin >> m2 >> a2;
        LL k1, k2;
        LL d = exgcd(m1, m2, k1, k2);
        if ((a2 - a1) % d){
            a1 = -1; //无解
            break;
        }
        k1 *= (a2 - a1) / d;
        LL t = m2 / d;
        k1 = (k1 % t + t) % t;
        a1 = k1 * m1 + a1;
        m1 = m1 * t;
    }
    cout << a1;
}
```

组合数

经典组合恒等式（上指标求和）：

$$\binom{a}{a}+\binom{a+1}{a}+\cdots+\binom{n}{a}=\binom{n+1}{a+1}. \tag{30}$$

组合恒等式：

$$\sum_{i=0}^{k} i\cdot\binom{n-i-1}{n-k-1}=\binom{n}{k-1}. \tag{31}$$

利用第二类斯特林数展开幂次：

$$i^k=\sum_{j=1}^{k} S(k,j)\binom{i}{j}j!, \tag{32}$$

其中 $S(k,j)$ 表示第二类斯特林数，$\binom{i}{j}$ 为组合数。

组合数恒等式：

$$\binom{n}{m}\binom{m}{r}=\binom{n}{r}\binom{n-r}{m-r}. \tag{33}$$

范德蒙德卷积公式：

$$\sum_{k=0}^{r}\binom{m}{k}\binom{n}{r-k}=\binom{m+n}{r},\qquad r=\min(n,m). \tag{34}$$

二项式定理：

$$\sum_{i=0}^{n}\binom{n}{i}X^i\ =\ (1+X)^n. \tag{35}$$

组合恒等式：

$$m\cdot\binom{n}{m}\ =\ n\cdot\binom{n-1}{m-1}. \tag{36}$$

组合恒等式：

$$\sum_{i=0}^{n} i\cdot\binom{n}{i}=n\cdot 2^{n-1}. \tag{37}$$

组合恒等式：

$$\sum_{i=0}^{n} i^2\binom{n}{i}=n(n+1)\,2^{n-2}. \tag{38}$$

组合恒等式：

$$\sum_{i=0}^{n}\binom{n}{i}^2=\binom{2n}{n}. \tag{39}$$

选定 n 个单调不增的数，且第一个数固定为 x，其方案数为：

$$\binom{n-1+x}{x}. \tag{40}$$

第二类斯特林数 $S(n,m)$ 的递推式：

$$S(n,m) = S(n-1,m-1) + m \cdot S(n-1,m). \tag{41}$$

n 个不一样的球放入 m 个一样的盒子中（非空）

其显式公式：

$$S(n,m) = \frac{1}{m!}\sum_{i=0}^{m}(-1)^i\binom{m}{i}(m-i)^n. \tag{42}$$

若盒子视为有标号，则方案数为：

$$m!\,S(n,m). \tag{43}$$

第一类斯特林数 $c(n,m)$ 的递推式：

$$c(n,m) = c(n-1,m-1) + (n-1)\,c(n-1,m). \tag{44}$$

其中 $c(n,m)$ 表示将 n 个互异元素划分为 m 个 **不区分的非空轮换** 的方案数。

卡特兰数 h_n 的递推式：

$$h_0 = 1, \qquad h_n = \sum_{i=0}^{n-1} h_i\,h_{n-1-i}, \quad n \geq 2. \tag{45}$$

其显式公式：

$$h_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{1}{n+1}\binom{2n}{n}. \tag{46}$$

卡特兰数 h_n 的前几项为：

$$h_0 = 1, \quad h_1 = 1, \quad h_2 = 2, \quad h_3 = 5, \quad h_4 = 14, \quad h_5 = 42, \quad h_6 = 132, \quad h_7 = 429, \quad h_8 = 1430, \quad h_9 = 4862, \quad h_{10} = 16796. \tag{47}$$

错排数（Derangements） D_n 的递推公式：

$$D_n = (n-1)\big(D_{n-1} + D_{n-2}\big), \qquad n \geq 2, \tag{48}$$

其中

$$D_0 = 1, \quad D_1 = 0. \tag{49}$$

错排数 D_n 的前几项为：

$$D_0 = 1, \quad D_1 = 0, \quad D_2 = 1, \quad D_3 = 2, \quad D_4 = 9, \quad D_5 = 44, \quad D_6 = 265, \quad D_7 = 1854, \quad D_8 = 14833, \quad D_9 = 133496, \quad D_{10} = 133$$

递推求组合数

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j <= i; j++) {
        if (!j || j == i) c[i][j] = 1;
        else c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % mod;
    }
}
```

预处理逆元求组合数

首先预处理出所有阶乘取模的余数 fact[N]，以及所有阶乘取模的逆元 infact[N], 如果取模的数是质数，可以用费马小定理求逆元

```
LL qmi(LL a, LL k, LL p) {
    LL res = 1;
    while (k) {
        if (k & 1) res = res * a % p;
        a = a * a % p;
        k >>= 1;
    }
    return res;
}
// 预处理阶乘的余数和阶乘逆元的余数
fact[0] = infact[0] = 1;
for (int i = 1; i < N; i++) {
    fact[i] = (LL)fact[i - 1] * i % mod;
    infact[i] = (LL)infact[i - 1] * qmi(i, mod - 2, mod) % mod;
}
// 线性预处理阶乘的余数和阶乘逆元的余数
fact[0] = infact[0] = 1;
for (int i = 1; i < N; i++) fact[i] = (LL)fact[i - 1] * i % mod;
infact[N - 1] = qmi(fact[N - 1], mod - 2, mod);
for (int i = N - 2; i >= 1; i--) infact[i] = infact[i + 1] * (i + 1) % mod;
```

lucas 定理

若 p 是质数，则对任意整数 $0 \leq m \leq n$，有

$$\binom{n}{m} \equiv \binom{n \bmod p}{m \bmod p} \cdot \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \pmod{p}. \tag{51}$$

```
LL qmi(LL a, LL k, LL p) {
    LL res = 1 % p;
    while (k) {
        if (k & 1) res = res * a % p;
        a = a * a % p;
        k >>= 1;
    }
    return res;
}
LL C(LL a, LL b, LL p) { // 通过定理求组合数 C(a, b)
    if (a < b) return 0;
    LL x = 1, y = 1;  // x 是分子，y 是分母
    for (LL i = a, j = 1; j <= b; i--, j++) {
        x = x * i % p;
        y = y * j % p;
    }
    return x * qmi(y, p - 2, p) % p;
}
LL lucas(LL a, LL b, LL p) {
    if (a < p && b < p) return C(a, b, p);
    return C(a % p, b % p, p) * lucas(a / p, b / p, p) % p;
}
```

分解质因数法求组合数

当我们需要求出组合数的真实值，而非对某个数的余数时，分解质因数的方式比较好用：

1. 筛法求出范围内的所有质数
2. 通过 C(a, b) = a! / b! / (a - b)! 这个公式求出每个质因子的次数。 n! 中 p 的次数是 n / p + n / p^2 + n / p^3 + ...
3. 用高精度乘法将所有质因子相乘

```
int primes[N], cnt;     // 存储所有质数
int sum[N];     // 存储每个质数的次数
bool st[N];     // 存储每个数是否已被筛掉
void get_primes(int n) {
    for (int i = 2; i <= n; i++) {
        if (!st[i]) primes[cnt++] = i;
        for (int j = 0; primes[j] <= n / i; j++) {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}
int get(int n, int p) {  // 求n! 中p出现的次数
    int res = 0;
    while (n) {
        res += n / p;
        n /= p;
    }
    return res;
}
vector<int> mul(vector<int> &a, int b) {
    vector<int> c;
    int t = 0;
    for (int i = 0; i < a.size(); i++) {
        t += a[i] * b;
        c.push_back(t % 10);
        t /= 10;
    }
    while (t) {
        c.push_back(t % 10);
        t /= 10;
    }
    return c;
}
void solve() {
    get_primes(a);  // 预处理范围内的所有质数
    for (int i = 0; i < cnt; i++) { // 求每个质因数的次数
        int p = primes[i];
        sum[i] = get(a, p) - get(b, p) - get(a - b, p);
    }
    vector<int> res;
    res.push_back(1);
    for (int i = 0; i < cnt; i++) {
        for (int j = 0; j < sum[i]; j++) {
            res = mul(res, primes[i]);
```

```
        }
    }
}
```

BSGS

普通BSGS

求 a 的 x 次方在模 p 意义下与 b 同余的最小的 x

普通 bsgs 中 a 与 p 互质，所以答案一定小于等于 phi(p)

```
int bsgs(int a, int b, int p) {
    if (1 % p == b % p) return 0;
    int k = sqrt(p) + 1;
    unordered_map<int, int> hh;
    for (int i = 0, j = b % p; i < k; i++) {
        hh[j] = i;
        j = (LL)j * a % p;
    }
    int ak = 1;
    for (int i = 1; i <= k; i++) ak = (LL)ak * a % p;   // 求 a^k
    for (int i = 1, j = 1 % p; i <= k; i++) {
        j = (LL)j * ak % p;
        if (hh.count(j)) return (LL)i * k - hh[j];
    }
    return -_MAX; // 无解
}
```

EXBSGS

扩展 bsgs 中不要求 a 与 p 互质

如果返回值小于 0 说明无解，可以通过将 -_MAX 设计的更低来确保正确，不过由于递归是 log 级别的，值只要小于 -log 就可以

```
int exgcd(int a, int b, int &x, int &y) {
    if (b == 0) {
        x = 1, y = 0;
        return a;
    }
    int d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
int exbsgs(int a, int b, int p) {
    b = (b % p + p) % p;
    if (1 % p == b % p) return 0;
    int x, y;
    int d = exgcd(a, p, x, y);
    if (d > 1) {
        if (b % d) return -1;
        exgcd(a / d, p / d, x, y);
        return exbsgs(a, (LL)b / d * x % (p / d), p / d) + 1;
    }
    return bsgs(a, b, p);
}
```

FFT

```
int n, m;
const long double PI = acosl(-1);
struct Complex {
    long double x, y;
    Complex operator+ (const Complex& t) const {
        return {x + t.x, y + t.y};
    }
    Complex operator- (const Complex& t) const {
        return {x - t.x, y - t.y};
    }
    Complex operator* (const Complex& t) const {
        return {x * t.x - y * t.y, x * t.y + y * t.x};
    }
} a[N], b[N];
int rev[N], bit, tot;
void fft(Complex a[], int inv) {
    for (int i = 0; i < tot; i++) {
        if (i < rev[i]) {
            swap(a[i], a[rev[i]]);
```

```
        }
    }
    for (int mid = 1; mid < tot; mid <<= 1) {
        auto w1 = Complex({cos(PI / mid), inv * sin(PI / mid)});
        for (int i = 0; i < tot; i += mid * 2) {
            auto wk = Complex({1, 0});
            for (int j = 0; j < mid; j++, wk = wk * w1) {
                auto x = a[i + j], y = wk * a[i + j + mid];
                a[i + j] = x + y, a[i + j + mid] = x - y;
            }
        }
    }
}
void solve() {
    cin >> n >> m;
    for (int i = 0; i <= n; i++) {
        cin >> a[i].x;
    }
    for (int i = 0; i <= m; i++) {
        cin >> b[i].x;
    }
    while ((1 << bit) < n + m + 1) bit++;
    tot = 1 << bit;
    for (int i = 0; i < tot; i++) {
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));
    }
    fft(a, 1), fft(b, 1); // 先正向 求点表示法
    for (int i = 0; i < tot; i++) a[i] = a[i] * b[i];
    fft(a, -1); // 逆向
    for (int i = 0; i <= n + m; i++) {
        cout << (int)(a[i].x / tot + 0.5) << ' ';
    }
}
```

FWT

```
void Or(ll *a, ll type) {  // 迭代实现，数常更小
    for (ll x = 2; x <= n; x <<= 1) {
        ll k = x >> 1;
        for (ll i = 0; i < n; i += x) {
            for (ll j = 0; j < k; j++) {
                (a[i + j + k] += a[i + j] * type) %= P;
            }
        }
    }
}

void And(ll *a, ll type) {
    for (ll x = 2; x <= n; x <<= 1) {
        ll k = x >> 1;
        for (ll i = 0; i < n; i += x) {
            for (ll j = 0; j < k; j++) {
                (a[i + j] += a[i + j + k] * type) %= P;
            }
        }
    }
}

void Xor(ll *a, ll type) {
    for (ll x = 2; x <= n; x <<= 1) {
        ll k = x >> 1;
        for (ll i = 0; i < n; i += x) {
            for (ll j = 0; j < k; j++) {
                (a[i + j] += a[i + j + k]) %= P;
                (a[i + j + k] = a[i + j] - a[i + j + k] * 2) %= P;
                (a[i + j] *= type) %= P;
                (a[i + j + k] *= type) %= P;
            }
        }
    }
}
```

Min-25筛

素数 0-2 次求和

```
struct Min25Pack {
    using i128 = __int128_t;
    long long MOD; // <0 => no mod
```

```
    long long n, sq;
    vector<int> primes;            // primes <= sqrt(n)
    vector<bool> isp;
    vector<long long> w;           // distinct floor(n/i)
    vector<int> id1, id2;          // index maps
    vector<long long> G0, G1, G2;  // block values after processing primes < current p
    Min25Pack(long long _n, long long _MOD = -1): MOD(_MOD), n(_n) {
        if (n < 2) { sq = 1; return; }
        sq = sqrtl(n);
        sieve_small();
        build_w();
        build_index();
        init_G();
        apply();
    }
    // ---------- math helpers ----------
    inline long long norm(long long x) const {
        if (MOD < 0) return x;
        x %= MOD; if (x < 0) x += MOD; return x;
    }
    inline long long add(long long a, long long b) const {
        if (MOD < 0) return a + b; a += b; if (a >= MOD) a -= MOD; return a;
    }
    inline long long sub(long long a, long long b) const {
        if (MOD < 0) return a - b; a -= b; if (a < 0) a += MOD; return a;
    }
    inline long long mul(long long a, long long b) const {
        if (MOD < 0) { return (long long)((i128)a * (i128)b); }
        return (long long)(((i128)a * (i128)b) % MOD);
    }
    // sum_{i=1}^x i^k helpers (safe, supports x up to ~1e12+)
    inline long long S0(long long x) const { return norm(x); }
    inline long long S1(long long x) const {
        if (MOD < 0) return (long long)((i128)x * (x + 1) / 2);
        return (long long)(((i128)x * (x + 1) / 2) % MOD);
    }
    inline long long S2(long long x) const {
        if (MOD < 0) return (long long)(((i128)x * (x + 1) * (2 * x + 1)) / 6);
        return (long long)((((i128)x * (x + 1) * (2 * x + 1)) / 6) % MOD);
    }
    // build small sieve <= sqrt(n)
    void sieve_small() {
        isp.assign((size_t)sq + 1, true);
        if (sq >= 0) isp[0] = false; if (sq >= 1) isp[1] = false;
        for (int i = 2; i <= sq; ++i) if (isp[i]) {
            primes.push_back(i);
            if ((long long)i * i <= sq)
                for (long long j = 1LL * i * i; j <= sq; j += i) isp[(size_t)j] = false;
        }
    }
    void build_w() {
        w.reserve((size_t)2 * sq + 5);
        for (long long l = 1, r; l <= n; l = r + 1) {
            r = n / (n / l);
            w.push_back(n / l);
        }
        // w 自然降序 (因 n/l 随 l 增减小)
    }
    void build_index() {
        id1.assign((size_t)sq + 1, -1);
        id2.assign((size_t)sq + 1, -1);
        for (int i = 0; i < (int)w.size(); ++i) {
            long long x = w[i];
            if (x <= sq) id1[x] = i; else id2[n / x] = i; // n/x<=sq
        }
    }
    inline int get_id(long long x) const {
        return (x <= sq) ? id1[x] : id2[n / x];
    }
    void init_G() {
        size_t m = w.size();
        G0.resize(m); G1.resize(m); G2.resize(m);
        for (size_t i = 0; i < m; ++i) {
            long long x = w[i];
            G0[i] = sub(S0(x), 1);   // Σ_{2..x} 1
            G1[i] = sub(S1(x), 1);   // Σ_{2..x} i
            G2[i] = sub(S2(x), 1);   // Σ_{2..x} i^2
        }
    }
    void apply() {
        if (n < 2) return;
```

```
        long long pre0 = 0; // count of processed primes
        long long pre1 = 0; // sum p
        long long pre2 = 0; // sum p^2
        for (size_t k = 0;k < primes.size(); ++k) {
            long long p = primes[k];
            long long p2 = p * (long long)p;
            if (p2 > n) break; // 无需再更新，因为条件 w[i]>=p2 才处理
            for (size_t i = 0; i < w.size() && w[i] >= p2; ++i) {
                int j = get_id(w[i] / p);
                // G0: count-like
                G0[i] = sub(G0[i], sub(G0[j], pre0));
                // G1: sum p    -> 减去 (G1[j]-pre1)*p
                G1[i] = sub(G1[i], mul(sub(G1[j], pre1), p));
                // G2: sum p^2  -> 减去 (G2[j]-pre2)*p^2
                G2[i] = sub(G2[i], mul(sub(G2[j], pre2), mul(p, p)));
            }
            // 更新已处理素数前缀
            pre0 = add(pre0, 1);
            pre1 = add(pre1, p);
            pre2 = add(pre2, mul(p, p));
        }
    }
    // 提取结果：Pk(n)
    long long prime_count() const { if (n < 2) return 0; return G0[get_id(n)]; }
    long long prime_sum()   const { if (n < 2) return 0; return G1[get_id(n)]; }
    long long prime2_sum()  const { if (n < 2) return 0; return G2[get_id(n)]; }
};
long long min25_pi(long long n) {
    Min25Pack M(n, -1);
    return M.prime_count();
}
long long min25_prime_sum(long long n) {
    Min25Pack M(n, -1);
    return M.prime_sum();
}
long long min25_prime2_sum(long long n) {
    Min25Pack M(n, -1);
    return M.prime2_sum();
}
long long min25_interval_count(long long L, long long R) {
    if (R < L) return 0;
    if (R < 2) return 0;
    if (L < 2) L = 2;
    return min25_pi(R) - min25_pi(L - 1);
}
long long min25_interval_sum(long long L, long long R) {
    if (R < L) return 0;
    if (R < 2) return 0;
    if (L < 2) L = 2;
    return min25_prime_sum(R) - min25_prime_sum(L - 1);
}
```

素数 0-8 次求和

```
template<int K_MAX>
struct Min25Multi {
    static_assert(K_MAX <= 8, "Faulhaber hardcoded to k<=8 in this header.");
    using i128 = __int128_t;
    long long n;         // query bound
    long long sq;        // floor(sqrt(n))
    long long MOD;       // <0 => no-mod
    // small sieve (primes up to sqrt(n))
    vector<int> primes;
    vector<bool> isp;
    // distinct floor(n / i) values
    vector<long long> w;
    vector<int> id1, id2;
    // block data: G[idx][k] holds current processed value over [2..w[idx]] for k-th power
    vector< array<long long, K_MAX + 1> > G;
    // ---- ctor ----
    Min25Multi(long long _n, long long _MOD = -1): n(_n), MOD(_MOD) {
        if (n < 2) {
            sq = 1;
            return; // trivial; all Pk=0
        }
        sq = (long long)floor(sqrtl((long double)n));
        sieve_small();
        build_w();
        build_index();
        init_G();
```

```
        apply();
    }
    /* ==================== arithmetic helpers ==================== */
    inline long long norm(long long x) const {
        if (MOD < 0) return x;
        x %= MOD;
        if (x < 0) x += MOD;
        return x;
    }
    inline long long add(long long a, long long b) const {
        if (MOD < 0) return a + b;
        a += b;
        if (a >= MOD) a -= MOD;
        if (a < 0) a += MOD; // in case of negative b
        return a;
    }
    inline long long sub(long long a, long long b) const {
        if (MOD < 0) return a - b;
        a -= b;
        if (a < 0) a += MOD;
        return a;
    }
    inline long long mul(long long a, long long b) const {
        if (MOD < 0) {
            return (long long)((i128)a * (i128)b);
        }
        return (long long)(((i128)a * (i128)b) % MOD);
    }
    /* ---- extended gcd mod inverse (works if gcd=1) ---- */
    static long long exgcd(long long a, long long b, long long &x, long long &y) {
        if (!b) { x = 1; y = 0; return a; }
        long long x1, y1;
        long long g = exgcd(b, a % b, x1, y1);
        x = y1;
        y = x1 - (a / b) * y1;
        return g;
    }
    long long mod_inv(long long a) const {
        a %= MOD; if (a < 0) a += MOD;
        long long x, y;
        long long g = exgcd(a, MOD, x, y);
        if (g != 1) {
            // not invertible; fallback (should not happen with prime MOD & known denominators)
            // return 0 to avoid crash; caller should ensure gcd=1.
            return 0;
        }
        x %= MOD;
        if (x < 0) x += MOD;
        return x;
    }
    /* ==================== Faulhaber sums S_k(x) (k<=8) ====================
     * Return sum_{i=1}^x i^k  (mod MOD if MOD>=0, else exact in 128 -> long long)
     * Formulas are integer; we do factorization to reduce overflow in no-mod branch.
     */
    inline long long S0(long long x) const { return norm(x); }
    inline long long S1(long long x) const {
        if (MOD < 0) return (long long)((i128)x * (x + 1) / 2);
        return (long long)(((i128)x * (x + 1) / 2) % MOD);
    }
    inline long long S2(long long x) const {
        if (MOD < 0) return (long long)(((i128)x * (x + 1) * (2 * x + 1)) / 6);
        i128 t = (i128)x * (x + 1) % MOD;
        t = t * ((2 * x + 1) % MOD) % MOD;
        return (long long)(t * mod_inv(6) % MOD);
    }
    inline long long S3(long long x) const {
        // (x^2 (x+1)^2)/4
        if (MOD < 0) {
            i128 t = (i128)x * x;
            t = t * (x + 1) * (x + 1);
            return (long long)(t / 4);
        }
        i128 t = (i128)x % MOD; t = t * t % MOD;
        i128 u = (i128)(x + 1) % MOD; u = u * u % MOD;
        t = t * u % MOD;
        return (long long)(t * mod_inv(4) % MOD);
    }
    inline long long S4(long long x) const {
        // x(x+1)(2x+1)(3x^2+3x-1)/30
        if (MOD < 0) {
            i128 a = x, b = x + 1, c = 2 * x + 1;
```

```
            i128 d = 3 * (i128)x * x + 3 * (i128)x - 1;
            i128 t = a * b; t %= (i128)9e18; // harmless hint; division safe below
            t = a * b * c; t = a * b * c; // re-eval clean
            t = a * b; t = t * c;
            t = t * d;
            return (long long)(t / 30);
        }
        long long a = x % MOD;
        long long b = (x + 1) % MOD;
        long long c = (2 * x + 1) % MOD;
        long long d = (((3 % MOD) * ((i128)x % MOD) * ((i128)x % MOD)) % MOD + ((3 % MOD) * a) % MOD + MOD - 1) % MOD;
        i128 t = a; t = t * b % MOD; t = t * c % MOD; t = t * d % MOD;
        return (long long)(t * mod_inv(30) % MOD);
    }
    inline long long S5(long long x) const {
        // x^2 (x+1)^2 (2x^2+2x-1)/12
        if (MOD < 0) {
            i128 a = x; a *= x;
            i128 b = x + 1; b *= b;
            i128 c = 2 * (i128)x * x + 2 * (i128)x - 1;
            i128 t = a * b; t *= c;
            return (long long)(t / 12);
        }
        long long a = (i128)x % MOD; a = (i128)a * a % MOD;
        long long b = (i128)(x + 1) % MOD; b = (i128)b * b % MOD;
        long long c = ((2 % MOD) * ((i128)x % MOD) * ((i128)x % MOD) % MOD + (2 % MOD) * ((i128)x % MOD) + MOD - 1) %
MOD;
        i128 t = a; t = t * b % MOD; t = t * c % MOD;
        return (long long)(t * mod_inv(12) % MOD);
    }
    inline long long S6(long long x) const {
        // x(x+1)(2x+1)(3x^4 +6x^3 -3x +1)/42
        if (MOD < 0) {
            i128 a = x, b = x + 1, c = 2 * x + 1;
            i128 x2 = (i128)x * x;
            i128 x3 = x2 * x;
            i128 x4 = x3 * x;
            i128 d = 3 * x4 + 6 * x3 - 3 * x + 1;
            i128 t = a * b; t *= c; t *= d;
            return (long long)(t / 42);
        }
        long long xm = x % MOD;
        long long a = xm;
        long long b = (x + 1) % MOD;
        long long c = (2 * x + 1) % MOD;
        i128 x2 = (i128)xm * xm % MOD;
        i128 x3 = x2 * xm % MOD;
        i128 x4 = x3 * xm % MOD;
        long long d = ((3 * x4) % MOD + (6 * x3) % MOD + MOD - (3 * xm) % MOD + 1) % MOD;
        i128 t = a; t = t * b % MOD; t = t * c % MOD; t = t * d % MOD;
        return (long long)(t * mod_inv(42) % MOD);
    }
    inline long long S7(long long x) const {
        // x^2 (x+1)^2 (3x^4 +6x^3 -x^2 -4x +2)/24
        if (MOD < 0) {
            i128 xx = x;
            i128 a = xx * xx;
            i128 b = (xx + 1) * (xx + 1);
            i128 x2 = xx * xx;
            i128 x3 = x2 * xx;
            i128 x4 = x3 * xx;
            i128 d = 3 * x4 + 6 * x3 - x2 - 4 * xx + 2;
            i128 t = a * b; t *= d;
            return (long long)(t / 24);
        }
        long long xm = x % MOD;
        long long a = (i128)xm * xm % MOD;
        long long b = (i128)(x + 1) % MOD; b = (i128)b * b % MOD;
        i128 x2 = (i128)xm * xm % MOD;
        i128 x3 = x2 * xm % MOD;
        i128 x4 = x3 * xm % MOD;
        long long d = ((3 * x4) % MOD + (6 * x3) % MOD + MOD - x2 + MOD - (4 * xm) % MOD + 2) % MOD;
        i128 t = a; t = t * b % MOD; t = t * d % MOD;
        return (long long)(t * mod_inv(24) % MOD);
    }
    inline long long S8(long long x) const {
        // x(x+1)(2x+1)(5x^6 +15x^5 -5x^4 -15x^3 +4x^2 +6x -1)/90
        if (MOD < 0) {
            i128 xx = x;
            i128 a = xx, b = xx + 1, c = 2 * xx + 1;
            i128 x2 = xx * xx;
```

```
            i128 x3 = x2 * xx;
            i128 x4 = x3 * xx;
            i128 x5 = x4 * xx;
            i128 x6 = x5 * xx;
            i128 d = 5 * x6 + 15 * x5 - 5 * x4 - 15 * x3 + 4 * x2 + 6 * xx - 1;
            i128 t = a * b; t *= c; t *= d;
            return (long long)(t / 90);
        }
        long long xm = x % MOD;
        long long a = xm;
        long long b = (x + 1) % MOD;
        long long c = (2 * x + 1) % MOD;
        i128 x2 = (i128)xm * xm % MOD;
        i128 x3 = x2 * xm % MOD;
        i128 x4 = x3 * xm % MOD;
        i128 x5 = x4 * xm % MOD;
        i128 x6 = x5 * xm % MOD;
        long long d = ((5 * x6) % MOD + (15 * x5) % MOD + MOD - (5 * x4) % MOD + MOD - (15 * x3) % MOD + (4 * x2) % MOD
+ (6 * xm) % MOD + MOD - 1) % MOD;
        i128 t = a; t = t * b % MOD; t = t * c % MOD; t = t * d % MOD;
        return (long long)(t * mod_inv(90) % MOD);
    }
    // dispatch
    inline long long sum_pow(int k, long long x) const {
        switch(k) {
            case 0: return S0(x);
            case 1: return S1(x);
            case 2: return S2(x);
            case 3: return S3(x);
            case 4: return S4(x);
            case 5: return S5(x);
            case 6: return S6(x);
            case 7: return S7(x);
            default: return S8(x);
        }
    }
    /* ==================== sieve <= sqrt(n) ==================== */
    void sieve_small() {
        isp.assign((size_t)sq + 1, true);
        if (sq >= 0) isp[0] = false;
        if (sq >= 1) isp[1] = false;
        for (int i = 2; i <= sq; ++i) if (isp[i]) {
            primes.push_back(i);
            if ((long long)i * i <= sq)
                for (long long j = 1LL * i * i; j <= sq; j += i) isp[(size_t)j] = false;
        }
    }
    /* ==================== build w: distinct floor(n/i) ==================== */
    void build_w() {
        w.reserve((size_t)2 * sq + 5);
        for (long long l = 1, r; l <= n; l = r + 1) {
            r = n / (n / l);
            w.push_back(n / l); // descending
        }
    }
    /* ==================== index maps ==================== */
    void build_index() {
        id1.assign((size_t)sq + 1, -1);
        id2.assign((size_t)sq + 1, -1);
        for (int i = 0; i < (int)w.size(); ++i) {
            long long x = w[i];
            if (x <= sq) id1[x] = i;
            else id2[n / x] = i; // n/x <= sq
        }
    }
    inline int get_id(long long x) const {
        return (x <= sq) ? id1[x] : id2[n / x];
    }
    /* ==================== init G ==================== */
    void init_G() {
        size_t m = w.size();
        G.resize(m);
        for (size_t i = 0; i < m; ++i) {
            long long x = w[i];
            for (int k = 0; k <= K_MAX; ++k) {
                long long S = sum_pow(k, x);  // Σ_{i=1}^x i^k
                G[i][k] = sub(S, 1);           // remove i=1
            }
        }
    }
    /* ==================== apply Min_25 ==================== */
```

```
    void apply() {
        if (n < 2) return;
        array<long long, K_MAX + 1> pre{};
        for (auto &v : pre) v = 0; // Σ processed primes^k
        for (size_t pi = 0; pi < primes.size(); ++pi) {
            long long p = primes[pi];
            long long p2 = p * (long long)p;
            if (p2 > n) break;
            // precompute p^k
            array<long long, K_MAX + 1> pk;
            pk[0] = 1;
            for (int k = 1; k <= K_MAX; ++k) pk[k] = mul(pk[k - 1], p);
            for (size_t i = 0; i < w.size() && w[i] >= p2; ++i) {
                int j = get_id(w[i] / p);
                for (int k = 0; k <= K_MAX; ++k) {
                    long long delta = sub(G[j][k], pre[k]);
                    G[i][k] = sub(G[i][k], mul(delta, pk[k]));
                }
            }
            // update prefix
            for (int k = 0;k <= K_MAX; ++k) pre[k] = add(pre[k], pk[k]);
        }
    }
    /* ==================== results ==================== */
    long long P(int k) const {
        if (n < 2) return 0;
        return G[get_id(n)][k];
    }
    // 128-bit version (no-mod only). If MOD>=0, call P(k).
    __int128 P128(int k) const {
        if (MOD >= 0) return (__int128)P(k);
        if (n < 2) return 0;
        // reinterpret long long as full; since we never modulo, G already stores full
        return (__int128)G[get_id(n)][k];
    }
    /* ==================== polynomial helper ==================== */
    long long poly_sum(const vector<long long>& coef) const {
        // coef[c] = coefficient of p^c
        long long ans = 0;
        int d = (int)coef.size() - 1;
        if (d > K_MAX) d = K_MAX; // ignore higher terms (or assert)
        for (int k = 0; k <= d; ++k) {
            if (coef[k] == 0) continue;
            ans = add(ans, mul(norm(coef[k]), P(k)));
        }
        return ans;
    }
};
using Multi = Min25Multi<0>;
long long min25_P_interval(long long L, long long R, int k, long long MOD = -1) {
    if (R < L || R < 2) return 0;
    if (L < 2) L = 2;
    Multi MR(R, MOD);
    Multi ML(L - 1, MOD);
    long long r = MR.P(k);
    long long l = ML.P(k);
    if (MOD < 0) return r - l;
    long long ans = r - l;
    if (ans < 0) ans += MOD;
    return ans;
}
```

SG函数

```
int h[N], e[M], ne[M], idx;
int sg[N]; // 每个点的 sg 函数值
int din[N], dout[N]; // 入度，出度
void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}
int getsg(int u) {
    if (sg[u] >= 0) return sg[u];
    unordered_map<int,int> hh;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        int tmp = getsg(j);
        hh[tmp] = 1;
    }
    for (int i = 0;; i++) {
        if (hh[i] == 0) {
```

```
            sg[u] = i;
            break;
        }
    }
    return sg[u];
}
void solve() {
    cin >> n >> m >> k;
    memset(h, -1, sizeof(h));
    memset(sg, -1, sizeof(sg));
    for (int i = 1; i <= m; i++) {
        int x, y;
        cin >> x >> y;
        add(x, y);
        din[y]++;
        dout[x]++;
    }
    vector<int> root, end; // 有向无环图可能有多个起点和终点
    for (int i = 1; i <= n; i++) {
        if (din[i] == 0) {
            root.push_back(i);
        }
        if (dout[i] == 0) {
            end.push_back(i);
        }
    }
    for (int i = 0; i < end.size(); i++) sg[end[i]] = 0;
    for (int i = 0; i < root.size(); i++) getsg(root[i]);
    int ans = 0;
    for (int i = 1; i <= k; i++) {
        int x;
        cin >> x;
        ans ^= sg[x];
    }
    if (ans == 0) cout << "lose" << '\n';
    else cout << "win" << '\n';
}
```

搜索

模拟退火

到平面 n 个点的最小距离

正解为三分套三分，函数有凸性

```
typedef pair<double, double> PDD;
int n;
PDD q[N];
double ans = 1e8;
double rand(double l, double r) { // 随机得到 l 到 r 中的一个数
    return (double)rand() / RAND_MAX * (r - l) + l;
}
double get_dist(PDD a, PDD b) {
    double dx = a.first - b.first;
    double dy = a.second - b.second;
    return sqrt(dx * dx + dy * dy);
}
double calc(PDD p) {
    double res = 0;
    for (int i = 1; i <= n; i++) res += get_dist(p, q[i]);
    ans = min(ans, res);
    return res;
}
void SA() {
    PDD cur = {rand(0, 10000), rand(0, 10000)};
    for (double t = 10000; t > 1e-4; t *= 0.99) { // t 是温度,可以控制步长和概率
        PDD np = {rand(cur.first - t, cur.first + t), rand(cur.second - t, cur.second + t)};
        double dt = calc(np) - calc(cur);
        if (exp(-dt / t) > rand(0, 1)) cur = np;   // 如果求最大值 if (exp(dt / t) < rand(0, 1))
    }
}
void solve() {
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> q[i].first >> q[i].second;
    for (int i = 0; i < 100; i++) SA(); // 模拟退火迭代次数
    // while ((double)clock() / CLOCKS_PER_SEC < 0.8) SA(); 卡时间
    cout << (LL)(ans + 0.5) << '\n'; // 四舍五入
}
```

Dancing_links

精确覆盖

选最少的行使得每列恰好有一个 1

选法作为行，限制作为列, N 为矩阵中 1 的数量 + 矩阵 max(行，列)

```
int n, m;
int l[N], r[N], u[N], d[N], s[N], row[N], col[N], idx; // 左右上下，每列有几个 1，每个节点在的行，列
int ans[N], top;
void init() {
    for (int i = 0; i <= m; i++) {
        l[i] = i - 1, r[i] = i + 1;
        u[i] = d[i] = i;
    }
    l[0] = m, r[m] = 0;
    idx = m + 1;
}
void add(int &hh, int &tt, int x, int y) {
    row[idx] = x, col[idx] = y, s[y]++;
    u[idx] = y, d[idx] = d[y], u[d[y]] = idx, d[y] = idx;
    r[hh] = l[tt] = idx, r[idx] = tt, l[idx] = hh;
    tt = idx++;
}
void remove(int p) {
    r[l[p]] = r[p], l[r[p]] = l[p];
    for (int i = d[p]; i != p; i = d[i]) {
        for (int j = r[i]; j != i; j = r[j]) {
            s[col[j]]--;
            u[d[j]] = u[j], d[u[j]] = d[j];
        }
    }
}
void resume(int p) {
    for (int i = u[p]; i != p; i = u[i]) {
        for (int j = l[i]; j != i; j = l[j]) {
            u[d[j]] = j, d[u[j]] = j;
            s[col[j]]++;
        }
    }
    r[l[p]] = p, l[r[p]] = p;
}
bool dfs() {
    if (!r[0]) return 1;
    int p = r[0];
    for (int i = r[0]; i; i = r[i]) {
        if (s[i] < s[p]) {
            p = i;
        }
    }
    remove(p);
    for (int i = d[p]; i != p; i = d[i]) {
        ans[++top] = row[i];
        for (int j = r[i]; j != i; j = r[j]) remove(col[j]);
        if (dfs()) return 1;
        for (int j = l[i]; j != i; j = l[j]) resume(col[j]);
        top--;
    }
    resume(p);
    return 0;
}
void solve() {
    cin >> n >> m;
    init();
    for (int i = 1; i <= n; i++) {
        int hh = idx, tt = idx;
        for (int j = 1; j <= m; j++) {
            int x;
            cin >> x;
            if (x) add(hh, tt, i, j);
        }
    }
    if (dfs()) {
        for (int i = 1; i <= top; i++) cout << ans[i] << ' ';
        cout << '\n';
    }
    else cout << "No Solution!" << '\n';
}
```

重复覆盖问题

```
int n, m;
int l[N], r[N], u[N], d[N], col[N], row[N], s[N], idx;
int ans[N];
bool st[110];
void init() {
    for (int i = 0; i <= m; i++) {
        l[i] = i - 1, r[i] = i + 1;
        col[i] = u[i] = d[i] = i;
        s[i] = 0;
    }
    l[0] = m, r[m] = 0;
    idx = m + 1;
}
void add(int& hh, int& tt, int x, int y) {
    row[idx] = x, col[idx] = y, s[y]++;
    u[idx] = y, d[idx] = d[y], u[d[y]] = idx, d[y] = idx;
    r[hh] = l[tt] = idx, r[idx] = tt, l[idx] = hh;
    tt = idx++;
}
int h() {
    int cnt = 0;
    memset(st, 0, sizeof(st));
    for (int i = r[0]; i; i = r[i]) {
        if (st[col[i]]) continue;
        cnt++;
        st[col[i]] = 1;
        for (int j = d[i]; j != i; j = d[j]) {
            for (int k = r[j]; k != j; k = r[k]) {
                st[col[k]] = 1;
            }
        }
    }
    return cnt;
}
void remove(int p) {
    for (int i = d[p]; i != p; i = d[i]) {
        r[l[i]] = r[i];
        l[r[i]] = l[i];
    }
}
void resume(int p) {
    for (int i = u[p]; i != p; i = u[i]) {
        r[l[i]] = i;
        l[r[i]] = i;
    }
}
bool dfs(int k, int depth) {
    if (k + h() > depth) return 0;
    if (!r[0]) return 1;
    int p = r[0];
    for (int i = r[0]; i; i = r[i]) {
        if (s[i] < s[p]) {
            p = i;
        }
    }
    for (int i = d[p]; i != p; i = d[i]) {
        ans[k] = row[i];
        remove(i);
        for (int j = r[i]; j != i; j = r[j]) remove(j);
        if (dfs(k + 1, depth)) return 1;
        for (int j = l[i]; j != i; j = l[j]) resume(j);
        resume(i);
    }
    return 0;
}
void solve() {
    cin >> n >> m;
    init();
    for (int i = 1; i <= n; i++) {
        int hh = idx, tt = idx;
        for (int j = 1; j <= m; j++) {
            int x;
            cin >> x;
            if (x) add(hh, tt, i, j);
        }
    }
    int depth = 0;
    while (!dfs(0, depth)) depth++;
    cout << depth << '\n';
```

```
    for (int i = 0; i < depth; i++) cout << ans[i] << ' ';
}
```

网络流

点只走一次 拆点 拆成的两个点之间的边容量为一

边只走一次 边容量设为一

最大流模板

EK 求最大流 O(nm^2)

```
int n, m, S, T;
int h[N], e[M], ne[M], idx;
LL f[M], d[N];
int pre[N];
bool vis[N];
void add(int a, int b, LL c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
bool bfs() {
    queue<int> q;
    memset(vis, 0, sizeof(vis));
    q.push(S), vis[S] = 1, d[S] = INF;
    while (q.size()) {
        auto t = q.front();
        q.pop();
        for (int i = h[t]; i != -1; i = ne[i]) {
            int ver = e[i];
            if (!vis[ver] && f[i]) {
                vis[ver] = 1;
                d[ver] = min(d[t], f[i]);
                pre[ver] = i;
                q.push(ver);
                if (ver == T) return 1;
            }
        }
    }
    return 0;
}
LL EK() {
    LL r = 0;
    while (bfs()) {
        r += d[T];
        for (int i = T; i != S; i = e[pre[i] ^ 1]) {
            f[pre[i]] -= d[T];
            f[pre[i] ^ 1] += d[T];
        }
    }
    return r;
}
void solve() {
    cin >> n >> m >> S >> T;
    memset(h, -1, sizeof(h));
    while (m--) {
        int a, b;
        LL c;
        cin >> a >> b >> c;
        add(a, b, c);
    }
    cout << EK() << '\n';
}
```

Dinic 求最大流 O(n^2m)

在单位容量的网络上，Dinic 算法的总时间复杂度是 O(m * min(m^(1/2), n^(2/3)))

对于二分图 Dinic 算法的总时间复杂度是 O(m * n^0.5)

```
int n, m, S, T;
int h[N], e[M], ne[M], idx;
LL f[M], d[N];
int cur[N];
void add(int a, int b, LL c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
```

```
bool bfs() {
    queue<int> q;
    memset(d, -1, sizeof(d));
    q.push(S), d[S] = 0, cur[S] = h[S];
    while (q.size()) {
        auto t = q.front();
        q.pop();
        for (int i = h[t]; i != -1; i = ne[i]) {
            int ver = e[i];
            if (d[ver] == -1 && f[i]) {
                d[ver] = d[t] + 1;
                cur[ver] = h[ver];
                q.push(ver);
                if (ver == T) return 1;
            }
        }
    }
    return 0;
}
LL find(int u, LL limit) {
    if (u == T) return limit;
    LL flow = 0;
    for (int i = cur[u]; i != -1 && flow < limit; i = ne[i]) {
        cur[u] = i;
        int ver = e[i];
        if (d[ver] == d[u] + 1 && f[i]) {
            LL t = find(ver, min(f[i], limit - flow));
            if (!t) d[ver] = -1;
            f[i] -= t, f[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}
LL dinic() {
    LL r = 0, flow;
    while (bfs()) {
        while (flow = find(S, INF)) r += flow;
    }
    return r;
}
void solve() {
    cin >> n >> m >> S >> T;
    memset(h, -1, sizeof(h));
    while (m--) {
        int a, b;
        LL c;
        cin >> a >> b >> c;
        add(a, b, c);
    }
    cout << dinic() << '\n';
}
```

费用流模板

最小费用最大流，只能在不存在负权回路时使用，最大费用最大流只需将边权取反最后将答案取反

EK 求费用流

O(nmf) f 是最大流

```
int n, m, S, T;
int h[N], e[M], f[M], w[M], ne[M], idx;
int d[N], pre[N], incf[N], st[N];
void add(int a, int b, int c, int d) {
    e[idx] = b, f[idx] = c, w[idx] = d, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, w[idx] = -d, ne[idx] = h[b], h[b] = idx++;
}
bool spfa() {
    queue<int> q;
    memset(d, 0x3f, sizeof(d));
    memset(incf, 0, sizeof(incf));
    memset(st, 0, sizeof(st));
    q.push(S), d[S] = 0, incf[S] = INF;
    while (q.size()) {
        auto t = q.front();
        q.pop();
        st[t] = 0;
        for (int i = h[t]; i != -1; i = ne[i]) {
            int ver = e[i];
            if (f[i] && d[ver] > d[t] + w[i]) {
```

```
                d[ver] = d[t] + w[i];
                pre[ver] = i;
                incf[ver] = min(f[i], incf[t]);
                if (!st[ver]) {
                    st[ver] = 1;
                    q.push(ver);
                }
            }
        }
    }
    return incf[T] > 0;
    // 可行费用流
    // return d[T] < 0;
}
void EK(int &flow, int &cost) {
    flow = cost = 0;
    while (spfa()) {
        int t = incf[T];
        flow += t, cost += t * d[T];
        for (int i = T; i != S; i = e[pre[i] ^ 1]) {
            f[pre[i]] -= t;
            f[pre[i] ^ 1] += t;
        }
    }
}
void solve() {
    cin >> n >> m >> S >> T;
    memset(h, -1, sizeof(h));
    while (m--) {
        int a, b, c, d;
        cin >> a >> b >> c >> d;
        add(a, b, c, d);
    }
    int flow, cost;
    EK(flow, cost);
    cout << flow << ' ' << cost << '\n';
}
```

原始对偶

O(mf logn) f 是最大流

注意更新范围

```
struct edge {
    int v, f, next;
    int c;
    // long double c;
} e[M];
struct node {
  int v, e;
} p[N];
struct mypair {
    int dis;
    // long double dis;
    int id;
    bool operator<(const mypair& a) const { return dis > a.dis; }
    mypair(int d, int x) { dis = d, id = x; }
    // mypair(long double d, int x) { dis = d, id = x; }
};
int head[N], vis[N];
int dis[N], h[N];
// long double dis[N], h[N];
int n, m, s, t, cnt = 1, maxf;
int minc;
// long double minc;
void addedge(int u, int v, int f, int c) { // or long double c
    e[++cnt].v = v;
    e[cnt].f = f;
    e[cnt].c = c;
    e[cnt].next = head[u];
    head[u] = cnt;
}
bool dijkstra() {
    priority_queue<mypair> q;
    for (int i = 1; i <= n; i++) dis[i] = INF; // 注意更新范围*********
    memset(vis, 0, sizeof(vis));
    dis[s] = 0;
    q.push(mypair(0, s));
    while (!q.empty()) {
```

```
        int u = q.top().id;
        q.pop();
        if (vis[u]) continue;
        vis[u] = 1;
        for (int i = head[u]; i; i = e[i].next) {
            int v = e[i].v, nc = e[i].c + h[u] - h[v];
            if (e[i].f && dis[v] > dis[u] + nc) {
                dis[v] = dis[u] + nc;
                p[v].v = u;
                p[v].e = i;
                if (!vis[v]) q.push(mypair(dis[v], v));
            }
        }
    }
    return dis[t] != INF;
}
void spfa() {
    queue<int> q;
    memset(h, 0x3f, sizeof(h));
    h[s] = 0, vis[s] = 1;
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        vis[u] = 0;
        for (int i = head[u]; i; i = e[i].next) {
            int v = e[i].v;
            if (e[i].f && h[v] > h[u] + e[i].c) {
                h[v] = h[u] + e[i].c;
                if (!vis[v]) {
                    vis[v] = 1;
                    q.push(v);
                }
            }
        }
    }
}
void work() {
    spfa();  // 先求出初始势能
    while (dijkstra()) {
        int minf = INF;
        for (int i = 1; i <= n; i++) h[i] += dis[i];  // 注意**********
        for (int i = t; i != s; i = p[i].v) minf = min(minf, e[p[i].e].f);
        for (int i = t; i != s; i = p[i].v) {
            e[p[i].e].f -= minf;
            e[p[i].e ^ 1].f += minf;
        }
        maxf += minf;
        minc += minf * h[t];
    }
}
void solve() {
    cin >> n >> m >> s >> t;
    for (int i = 1; i <= m; i++) {
        int u, v, f, c;
        cin >> u >> v >> f >> c;
        addedge(u, v, f, c);
        addedge(v, u, 0, -c);
    }
    work();
    cout << maxf << ' ' << minc << '\n';
}
```

点覆盖、独立集

源点向起点连权值边，终点向汇点连权值边，内部边容量为 INF

最大权独立集 = 总权值 - 最小权点覆盖

```
int n, m, S, T;
int h[N], e[M], f[M], ne[M], idx;
int d[N], cur[N];
bool st[N];
void dfs(int u) { // 求方案
    st[u] = 1;
    for (int i = h[u]; i != -1; i = ne[i]) {
        if (f[i] && !st[e[i]]) {
            dfs(e[i]);
        }
    }
```

```
}
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    cin >> n >> m;
    S = 0, T = n * 2 + 1;
    memset(h, -1, sizeof(h));
    for (int i = 1; i <= n; i++) {
        int w;
        cin >> w;
        add(S, i, w);
    }
    for (int i = 1; i <= n; i++) {
        int w;
        cin >> w;
        add(n + i, T, w);
    }
    while (m--) {
        int a, b;
        cin >> a >> b;
        add(b, n + a, INF);
    }
    cout << dinic() << '\n';
    dfs(S);
    int cnt = 0;
    for (int i = 0; i < idx; i += 2) {
        int a = e[i ^ 1], b = e[i];
        if (st[a] && !st[b]) cnt++;
    }
    cout << cnt << '\n';
    for (int i = 0; i < idx; i += 2) {
        int a = e[i ^ 1], b = e[i];
        if (st[a] && !st[b]) {
            if (a == S) cout << b << " +" << '\n';
        }
    }
    for (int i = 0; i < idx; i += 2) {
        int a = e[i ^ 1], b = e[i];
        if (st[a] && !st[b]) {
            if (b == T) cout << a - n << " -" << '\n';
        }
    }
}
```

点连通度

拆点后枚举源点和汇点跑最小割

```
int n, m, S, T;
int h[N], e[M], f[M], ne[M], idx;
int d[N], cur[N];
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    cin >> n >> m;
    memset(h, -1, sizeof(h));
    for (int i = 0; i < n; i++) add(i, n + i, 1);
    while (m--) {
        int x, y;
        cin >> x >> y;
        add(n + x, y, INF), add(n + y, x, INF);
    }
    int res = n;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            S = n + i, T = j;
            for (int k = 0; k < idx; k += 2) {
                f[k] += f[k ^ 1];
                f[k ^ 1] = 0;
            }
            res = min(res, dinic());
        }
    }
    cout << res << '\n';
}
```

二分图匹配和方案

二分图最大匹配

O(mn^0.5)

```
int n, m, S, T;
int h[N], e[M], f[M], ne[M], idx;
int d[N], cur[N];
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    cin >> m >> n;
    S = 0, T = n + 1;
    memset(h, -1, sizeof(h));
    for (int i = 1; i <= m; i++) add(S, i, 1);
    for (int i = m + 1; i <= n; i++) add(i, T, 1);
    int a, b;
    while (cin >> a >> b, a != -1) add(a, b, 1);
    cout << dinic() << '\n';
    for (int i = 0; i < idx; i += 2) {
        if (e[i] > m && e[i] <= n && !f[i]) {
            cout << e[i ^ 1] << ' ' << e[i] << '\n';
        }
    }
}
```

二分图多重匹配

允许每个点与多条边匹配，但有容量限制

```
int m, n, S, T;
int h[N], e[M], ne[M], f[M], idx;
int d[N], cur[N];
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    cin >> m >> n;
    S = 0, T = m + n + 1;
    memset(h, -1, sizeof(h));
    int tot = 0;
    for (int i = 1; i <= m; i++) {
        int c;
        cin >> c;
        add(S, i, c);
        tot += c;
    }
    for (int i = 1; i <= n; i++) {
        int c;
        cin >> c;
        add(m + i, T, c);
    }
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) add(i, m + j, 1);
    }
    if (dinic() != tot) cout << 0 << '\n';
    else {
        cout << 1 << '\n';
        for (int i = 1; i <= m; i++) {
            for (int j = h[i]; j != -1; j = ne[j]) {
                if (e[j] > m && e[j] <= m + n && !f[j]) {
                    cout << e[j] - m << ' ';
                }
            }
            cout << '\n';
        }
    }
}
```

二分图最大权匹配

不仅要找最多的匹配，还要让匹配边的权值和最大

```
int n, S, T;
int h[N], e[M], f[M], w[M], ne[M], idx;
```

```
int d[N], pre[N], incf[N];
bool st[N];
void add(int a, int b, int c, int d) {
    e[idx] = b, f[idx] = c, w[idx] = d, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, w[idx] = -d, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    cin >> n;
    S = 0, T = n * 2 + 1;
    memset(h, -1, sizeof(h));
    for (int i = 1; i <= n; i++) {
        add(S, i, 1, 0);
        add(n + i, T, 1, 0);
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            int c;
            cin >> c;
            add(i, n + j, 1, c);
        }
    }
    int ansmin, ansmax;
    int flow, cost;
    EK(flow, cost);
    ansmin = cost;
    for (int i = 0; i < idx; i += 2) {
        f[i] += f[i ^ 1], f[i ^ 1] = 0;
        w[i] = -w[i], w[i ^ 1] = -w[i ^ 1];
    }
    EK(flow, cost);
    ansmax = -cost;
    cout << ansmin << '\n';
    cout << ansmax << '\n';
}
```

多源汇最大流

建超级源点和汇点，所有超级源点向所有源点连边，所有汇点向超级汇点连边

```
int n, m, S, T;
int h[N], e[M], f[M], ne[M], idx;
int d[N], cur[N];
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    int sc, tc;
    cin >> n >> m >> sc >> tc;
    S = 0, T = n + 1;
    memset(h, -1, sizeof(h));
    while (sc--) {
        int x;
        cin >> x;
        add(S, x, INF);
    }
    while (tc--) {
        int x;
        cin >> x;
        add(x, T, INF);
    }
    while (m--) {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c);
    }
    cout << dinic() << '\n';
}
```

多源汇费用流

```
int n, m, S, T;
int h[N], e[M], f[M], w[M], ne[M], idx;
int d[N], pre[N], incf[N];
bool st[N];
void add(int a, int b, int c, int d) {
    e[idx] = b, f[idx] = c, w[idx] = d, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, w[idx] = -d, ne[idx] = h[b], h[b] = idx++;
}
```

```
void solve() {
    cin >> m >> n;
    S = 0, T = m + n + 1;
    memset(h, -1, sizeof(h));
    for (int i = 1; i <= m; i++) {
        int a;
        cin >> a;
        add(S, i, a, 0);
    }
    for (int i = 1; i <= n; i++) {
        int a;
        cin >> a;
        add(m + i, T, a, 0);
    }
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            int c;
            cin >> c;
            add(i, m + j, INF, c);
        }
    }
    int flow, cost;
    EK(flow, cost);
    cout << cost << '\n'; // 最小费用

    for (int i = 0; i < idx; i += 2) {
        f[i] += f[i ^ 1];
        f[i ^ 1] = 0;
        w[i] = -w[i], w[i ^ 1] = -w[i ^ 1];
    }
    EK(flow, cost);
    cout << -cost << '\n'; // 最大费用
}
```

上下界可行流

无源汇上下界可行流

```
int n, m, S, T;
int h[N], e[M], f[M], l[M], ne[M], idx;
int d[N], cur[N], A[N];
void add(int a, int b, int c, int d) { // c:下界, d:上界
    e[idx] = b, f[idx] = d - c, l[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    cin >> n >> m;
    S = 0, T = n + 1;
    memset(h, -1, sizeof(h));
    for (int i = 0; i < m; i++) {
        int a, b, c, d;
        cin >> a >> b >> c >> d;
        add(a, b, c, d);
        A[a] -= c, A[b] += c;
    }
    int tot = 0;
    for (int i = 1; i <= n; i++) {
        if (A[i] > 0) {
            add(S, i, 0, A[i]);
            tot += A[i];
        }
        else if (A[i] < 0) add(i, T, 0, -A[i]);
    }
    if (dinic() != tot) cout << "NO" << '\n';
    else {
        cout << "YES" << '\n';
        for (int i = 0; i < m * 2; i += 2) cout << f[i ^ 1] + l[i] << '\n';
    }
}
```

有源汇上下界最大流

```
int n, m, S, T;
int h[N], e[M], f[M], ne[M], idx;
int d[N], cur[N], A[N];
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
```

```
void solve() {
    int s, t;
    cin >> n >> m >> s >> t;
    S = 0, T = n + 1;
    memset(h, -1, sizeof(h));
    for (int i = 0; i < m; i++) {
        int a, b, c, d;
        cin >> a >> b >> c >> d;
        add(a, b, d - c);
        A[a] -= c, A[b] += c;
    }
    int tot = 0;
    for (int i = 1; i <= n; i++) {
        if (A[i] > 0) {
            add(S, i, A[i]);
            tot += A[i];
        }
        else if (A[i] < 0) add(i, T, -A[i]);
    }
    add(t, s, INF);
    if (dinic() < tot) cout << "No Solution" << '\n';
    else {
        int res = f[idx - 1];
        S = s, T = t;
        f[idx - 1] = f[idx - 2] = 0;
        cout << res + dinic() << '\n';
    }
}
```

有源汇上下界最小流

```
int n, m, S, T;
int h[N], e[M], f[M], ne[M], idx;
int d[N], cur[N], A[N];
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    int s, t;
    cin >> n >> m >> s >> t;
    S = 0, T = n + 1;
    memset(h, -1, sizeof(h));
    for (int i = 0; i < m; i++) {
        int a, b, c, d;
        cin >> a >> b >> c >> d;
        add(a, b, d - c);
        A[a] -= c, A[b] += c;
    }
    int tot = 0;
    for (int i = 1; i <= n; i++) {
        if (A[i] > 0) {
            add(S, i, A[i]);
            tot += A[i];
        }
        else if (A[i] < 0) add(i, T, -A[i]);
    }
    add(t, s, INF);
    if (dinic() < tot) cout << "No Solution" << '\n';
    else {
        int res = f[idx - 1];
        S = t, T = s;
        f[idx - 1] = f[idx - 2] = 0;
        cout << res - dinic() << '\n';
    }
}
```

最大流关键边

求关键边数量

```
int n, m, S, T;
int h[N], e[M], f[M], ne[M], idx;
int d[N], cur[N];
bool vis_s[N], vis_t[N];
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
```

```
void dfs(int u, bool st[], int t) {
    st[u] = 1;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = i ^ t;
        int ver = e[i];
        if (f[j] && !st[ver]) dfs(ver, st, t);
    }
}
void solve() {
    cin >> n >> m;
    S = 0, T = n - 1;
    memset(h, -1, sizeof(h));
    for (int i = 0; i < m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c);
    }
    dinic();
    dfs(S, vis_s, 0);
    dfs(T, vis_t, 1);
    int res = 0;
    for (int i = 0; i < m * 2; i += 2) {
        if (!f[i] && vis_s[e[i ^ 1]] && vis_t[e[i]]) {
            res++;
        }
    }
    cout << res << '\n';
}
```

最大密度子图

边的数量 (E + V * 2) * 2

无点权无边权

```
int n, m, S, T;
int h[N], e[M], ne[M], idx;
double f[M];
int d[N], cur[N];
int dg[N];
struct Edge {
    int a, b;
}edges[M];
int ans;
bool st[N];
void add(int a, int b, double c1, double c2) {
    e[idx] = b, f[idx] = c1, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = c2, ne[idx] = h[b], h[b] = idx++;
}
bool bfs() {
    queue<int> q;
    memset(d, -1, sizeof(d));
    q.push(S), d[S] = 0, cur[S] = h[S];
    while (q.size()) {
        auto t = q.front();
        q.pop();
        for (int i = h[t]; i != -1; i = ne[i]) {
            int ver = e[i];
            if (d[ver] == -1 && f[i] > 0) {
                d[ver] = d[t] + 1;
                cur[ver] = h[ver];
                q.push(ver);
                if (ver == T) return 1;
            }
        }
    }
    return 0;
}
double find(int u, double limit) {
    if (u == T) return limit;
    double flow = 0;
    for (int i = cur[u]; i != -1 && flow < limit; i = ne[i]) {
        cur[u] = i;
        int ver = e[i];
        if (d[ver] == d[u] + 1 && f[i] > 0) {
            double t = find(ver, min(f[i], limit - flow));
            if (t < eps) d[ver] = -1;
            f[i] -= t, f[i ^ 1] += t, flow += t;
        }
    }
```

```
    return flow;
}
void build(double g) {
    memset(h, -1, sizeof(h));
    idx = 0;
    for (int i = 0; i < m; i++) add(edges[i].a, edges[i].b, 1, 1);
    for (int i = 1; i <= n; i++) {
        add(S, i, m, 0);
        add(i, T, m + 2 * g - dg[i], 0);
    }
}
double dinic(double g) {
    build(g);
    double r = 0, flow;
    while (bfs()) while (flow = find(S, INF)) r += flow;
    return r;
}
void dfs(int u) {
    st[u] = 1;
    if (u != S) ans++;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int ver = e[i];
        if (!st[ver] && f[i] > 0) dfs(ver);
    }
}
void solve() {
    cin >> n >> m;
    S = 0, T = n + 1;
    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        dg[a]++, dg[b]++;
        edges[i] = {a, b};
    }
    double l = 0, r = m;
    while (r - l > eps) {
        double mid = (l + r) / 2;
        double t = dinic(mid);
        if (m * n - t > 0) l = mid;
        else r = mid;
    }
    dinic(l);
    dfs(S);
    if (!ans) {
        cout << 1 << '\n';
        cout << 1 << '\n';
        return;
    }
    cout << ans << '\n';
    for (int i = 1; i <= n; i++) {
        if (st[i]) cout << i << '\n';
    }
}
```

最大权闭合子图

最大权为 正权和 - 最小割

源点向正权点连权值边，负权点向汇点连权值的绝对值，有向图原来的边的权值为无限大

```
int n, m, S, T;
int h[N], e[M], f[M], ne[M], idx;
int d[N], cur[N];
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void dfs(int u) {
    st[u] = 1;
    for (int i = h[u]; i != -1; i = ne[i]) {
        if (!st[e[i]] && f[i]) {
            dfs(e[i]);
        }
    }
}
void solve() {
    cin >> n >> m;
    S = 0, T = n + m + 1;
    memset(h, -1, sizeof(h));
    for (int i = 1; i <= n; i++) {
```

```
        int p;
        cin >> p;
        add(m + i, T, p);
    }
    int tot = 0;
    for (int i = 1; i <= m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        add(S, i, c);
        add(i, m + a, INF);
        add(i, m + b, INF);
        tot += c;
    }
    cout << tot - dinic() << '\n';
    dfs(S); // 求方案
    for (int i = 1; i <= m; i++) {
        if (st[i]) cout << i << ' ';
    }
    cout << '\n';
    for (int i = m + 1; i <= m + n; i++) {
        if (st[i]) cout << i - m << ' ';
    }
}
```

网络流

点只走一次 拆点 拆成的两个点之间的边容量为一

边只走一次 边容量设为一

最大流模板

EK 求最大流 O(nm^2)

```
int n, m, S, T;
int h[N], e[M], ne[M], idx;
LL f[M], d[N];
int pre[N];
bool vis[N];
void add(int a, int b, LL c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
bool bfs() {
    queue<int> q;
    memset(vis, 0, sizeof(vis));
    q.push(S), vis[S] = 1, d[S] = INF;
    while (q.size()) {
        auto t = q.front();
        q.pop();
        for (int i = h[t]; i != -1; i = ne[i]) {
            int ver = e[i];
            if (!vis[ver] && f[i]) {
                vis[ver] = 1;
                d[ver] = min(d[t], f[i]);
                pre[ver] = i;
                q.push(ver);
                if (ver == T) return 1;
            }
        }
    }
    return 0;
}
LL EK() {
    LL r = 0;
    while (bfs()) {
        r += d[T];
        for (int i = T; i != S; i = e[pre[i] ^ 1]) {
            f[pre[i]] -= d[T];
            f[pre[i] ^ 1] += d[T];
        }
    }
    return r;
}
void solve() {
    cin >> n >> m >> S >> T;
    memset(h, -1, sizeof(h));
    while (m--) {
        int a, b;
        LL c;
```

```
        cin >> a >> b >> c;
        add(a, b, c);
    }
    cout << EK() << '\n';
}
```

Dinic 求最大流 O(n^2m)

在单位容量的网络上，Dinic 算法的总时间复杂度是 O(m * min(m^(1/2), n^(2/3)))

对于二分图 Dinic 算法的总时间复杂度是 O(m * n^0.5)

```
int n, m, S, T;
int h[N], e[M], ne[M], idx;
LL f[M], d[N];
int cur[N];
void add(int a, int b, LL c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
bool bfs() {
    queue<int> q;
    memset(d, -1, sizeof(d));
    q.push(S), d[S] = 0, cur[S] = h[S];
    while (q.size()) {
        auto t = q.front();
        q.pop();
        for (int i = h[t]; i != -1; i = ne[i]) {
            int ver = e[i];
            if (d[ver] == -1 && f[i]) {
                d[ver] = d[t] + 1;
                cur[ver] = h[ver];
                q.push(ver);
                if (ver == T) return 1;
            }
        }
    }
    return 0;
}
LL find(int u, LL limit) {
    if (u == T) return limit;
    LL flow = 0;
    for (int i = cur[u]; i != -1 && flow < limit; i = ne[i]) {
        cur[u] = i;
        int ver = e[i];
        if (d[ver] == d[u] + 1 && f[i]) {
            LL t = find(ver, min(f[i], limit - flow));
            if (!t) d[ver] = -1;
            f[i] -= t, f[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}
LL dinic() {
    LL r = 0, flow;
    while (bfs()) {
        while (flow = find(S, INF)) r += flow;
    }
    return r;
}
void solve() {
    cin >> n >> m >> S >> T;
    memset(h, -1, sizeof(h));
    while (m--) {
        int a, b;
        LL c;
        cin >> a >> b >> c;
        add(a, b, c);
    }
    cout << dinic() << '\n';
}
```

费用流模板

最小费用最大流，只能在不存在负权回路时使用，最大费用最大流只需将边权取反最后将答案取反

EK 求费用流

O(nmf) f 是最大流

```
int n, m, S, T;
int h[N], e[M], f[M], w[M], ne[M], idx;
int d[N], pre[N], incf[N], st[N];
void add(int a, int b, int c, int d) {
    e[idx] = b, f[idx] = c, w[idx] = d, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, w[idx] = -d, ne[idx] = h[b], h[b] = idx++;
}
bool spfa() {
    queue<int> q;
    memset(d, 0x3f, sizeof(d));
    memset(incf, 0, sizeof(incf));
    memset(st, 0, sizeof(st));
    q.push(S), d[S] = 0, incf[S] = INF;
    while (q.size()) {
        auto t = q.front();
        q.pop();
        st[t] = 0;
        for (int i = h[t]; i != -1; i = ne[i]) {
            int ver = e[i];
            if (f[i] && d[ver] > d[t] + w[i]) {
                d[ver] = d[t] + w[i];
                pre[ver] = i;
                incf[ver] = min(f[i], incf[t]);
                if (!st[ver]) {
                    st[ver] = 1;
                    q.push(ver);
                }
            }
        }
    }
    return incf[T] > 0;
    // 可行费用流
    // return d[T] < 0;
}
void EK(int &flow, int &cost) {
    flow = cost = 0;
    while (spfa()) {
        int t = incf[T];
        flow += t, cost += t * d[T];
        for (int i = T; i != S; i = e[pre[i] ^ 1]) {
            f[pre[i]] -= t;
            f[pre[i] ^ 1] += t;
        }
    }
}
void solve() {
    cin >> n >> m >> S >> T;
    memset(h, -1, sizeof(h));
    while (m--) {
        int a, b, c, d;
        cin >> a >> b >> c >> d;
        add(a, b, c, d);
    }
    int flow, cost;
    EK(flow, cost);
    cout << flow << ' ' << cost << '\n';
}
```

原始对偶

O(mf logn) f 是最大流

注意更新范围

```
struct edge {
    int v, f, next;
    int c;
    // long double c;
} e[M];
struct node {
  int v, e;
} p[N];
struct mypair {
    int dis;
    // long double dis;
    int id;
    bool operator<(const mypair& a) const { return dis > a.dis; }
    mypair(int d, int x) { dis = d, id = x; }
    // mypair(long double d, int x) { dis = d, id = x; }
};
```

```
int head[N], vis[N];
int dis[N], h[N];
// long double dis[N], h[N];
int n, m, s, t, cnt = 1, maxf;
int minc;
// long double minc;
void addedge(int u, int v, int f, int c) { // or long double c
    e[++cnt].v = v;
    e[cnt].f = f;
    e[cnt].c = c;
    e[cnt].next = head[u];
    head[u] = cnt;
}
bool dijkstra() {
    priority_queue<mypair> q;
    for (int i = 1; i <= n; i++) dis[i] = INF; // 注意更新范围*********
    memset(vis, 0, sizeof(vis));
    dis[s] = 0;
    q.push(mypair(0, s));
    while (!q.empty()) {
        int u = q.top().id;
        q.pop();
        if (vis[u]) continue;
        vis[u] = 1;
        for (int i = head[u]; i; i = e[i].next) {
            int v = e[i].v, nc = e[i].c + h[u] - h[v];
            if (e[i].f && dis[v] > dis[u] + nc) {
                dis[v] = dis[u] + nc;
                p[v].v = u;
                p[v].e = i;
                if (!vis[v]) q.push(mypair(dis[v], v));
            }
        }
    }
    return dis[t] != INF;
}
void spfa() {
    queue<int> q;
    memset(h, 0x3f, sizeof(h));
    h[s] = 0, vis[s] = 1;
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        vis[u] = 0;
        for (int i = head[u]; i; i = e[i].next) {
            int v = e[i].v;
            if (e[i].f && h[v] > h[u] + e[i].c) {
                h[v] = h[u] + e[i].c;
                if (!vis[v]) {
                    vis[v] = 1;
                    q.push(v);
                }
            }
        }
    }
}
void work() {
    spfa();  // 先求出初始势能
    while (dijkstra()) {
        int minf = INF;
        for (int i = 1; i <= n; i++) h[i] += dis[i];  // 注意**********
        for (int i = t; i != s; i = p[i].v) minf = min(minf, e[p[i].e].f);
        for (int i = t; i != s; i = p[i].v) {
            e[p[i].e].f -= minf;
            e[p[i].e ^ 1].f += minf;
        }
        maxf += minf;
        minc += minf * h[t];
    }
}
void solve() {
    cin >> n >> m >> s >> t;
    for (int i = 1; i <= m; i++) {
        int u, v, f, c;
        cin >> u >> v >> f >> c;
        addedge(u, v, f, c);
        addedge(v, u, 0, -c);
    }
    work();
    cout << maxf << ' ' << minc << '\n';
```

```
}
```

点覆盖、独立集

源点向起点连权值边，终点向汇点连权值边，内部边容量为 INF

最大权独立集 = 总权值 - 最小权点覆盖

```
int n, m, S, T;
int h[N], e[M], f[M], ne[M], idx;
int d[N], cur[N];
bool st[N];
void dfs(int u) { // 求方案
    st[u] = 1;
    for (int i = h[u]; i != -1; i = ne[i]) {
        if (f[i] && !st[e[i]]) {
            dfs(e[i]);
        }
    }
}
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    cin >> n >> m;
    S = 0, T = n * 2 + 1;
    memset(h, -1, sizeof(h));
    for (int i = 1; i <= n; i++) {
        int w;
        cin >> w;
        add(S, i, w);
    }
    for (int i = 1; i <= n; i++) {
        int w;
        cin >> w;
        add(n + i, T, w);
    }
    while (m--) {
        int a, b;
        cin >> a >> b;
        add(b, n + a, INF);
    }
    cout << dinic() << '\n';
    dfs(S);
    int cnt = 0;
    for (int i = 0; i < idx; i += 2) {
        int a = e[i ^ 1], b = e[i];
        if (st[a] && !st[b]) cnt++;
    }
    cout << cnt << '\n';
    for (int i = 0; i < idx; i += 2) {
        int a = e[i ^ 1], b = e[i];
        if (st[a] && !st[b]) {
            if (a == S) cout << b << " +" << '\n';
        }
    }
    for (int i = 0; i < idx; i += 2) {
        int a = e[i ^ 1], b = e[i];
        if (st[a] && !st[b]) {
            if (b == T) cout << a - n << " -" << '\n';
        }
    }
}
```

点连通度

拆点后枚举源点和汇点跑最小割

```
int n, m, S, T;
int h[N], e[M], f[M], ne[M], idx;
int d[N], cur[N];
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    cin >> n >> m;
    memset(h, -1, sizeof(h));
```

```
    for (int i = 0; i < n; i++) add(i, n + i, 1);
    while (m--) {
        int x, y;
        cin >> x >> y;
        add(n + x, y, INF), add(n + y, x, INF);
    }
    int res = n;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            S = n + i, T = j;
            for (int k = 0; k < idx; k += 2) {
                f[k] += f[k ^ 1];
                f[k ^ 1] = 0;
            }
            res = min(res, dinic());
        }
    }
    cout << res << '\n';
}
```

二分图匹配和方案

二分图最大匹配

O(mn^0.5)

```
int n, m, S, T;
int h[N], e[M], f[M], ne[M], idx;
int d[N], cur[N];
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    cin >> m >> n;
    S = 0, T = n + 1;
    memset(h, -1, sizeof(h));
    for (int i = 1; i <= m; i++) add(S, i, 1);
    for (int i = m + 1; i <= n; i++) add(i, T, 1);
    int a, b;
    while (cin >> a >> b, a != -1) add(a, b, 1);
    cout << dinic() << '\n';
    for (int i = 0; i < idx; i += 2) {
        if (e[i] > m && e[i] <= n && !f[i]) {
            cout << e[i ^ 1] << ' ' << e[i] << '\n';
        }
    }
}
```

二分图多重匹配

允许每个点与多条边匹配，但有容量限制

```
int m, n, S, T;
int h[N], e[M], ne[M], f[M], idx;
int d[N], cur[N];
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    cin >> m >> n;
    S = 0, T = m + n + 1;
    memset(h, -1, sizeof(h));
    int tot = 0;
    for (int i = 1; i <= m; i++) {
        int c;
        cin >> c;
        add(S, i, c);
        tot += c;
    }
    for (int i = 1; i <= n; i++) {
        int c;
        cin >> c;
        add(m + i, T, c);
    }
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) add(i, m + j, 1);
    }
    if (dinic() != tot) cout << 0 << '\n';
```

```
    else {
        cout << 1 << '\n';
        for (int i = 1; i <= m; i++) {
            for (int j = h[i]; j != -1; j = ne[j]) {
                if (e[j] > m && e[j] <= m + n && !f[j]) {
                    cout << e[j] - m << ' ';
                }
            }
            cout << '\n';
        }
    }
}
```

二分图最大权匹配

不仅要找最多的匹配，还要让匹配边的权值和最大

```
int n, S, T;
int h[N], e[M], f[M], w[M], ne[M], idx;
int d[N], pre[N], incf[N];
bool st[N];
void add(int a, int b, int c, int d) {
    e[idx] = b, f[idx] = c, w[idx] = d, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, w[idx] = -d, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    cin >> n;
    S = 0, T = n * 2 + 1;
    memset(h, -1, sizeof(h));
    for (int i = 1; i <= n; i++) {
        add(S, i, 1, 0);
        add(n + i, T, 1, 0);
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            int c;
            cin >> c;
            add(i, n + j, 1, c);
        }
    }
    int ansmin, ansmax;
    int flow, cost;
    EK(flow, cost);
    ansmin = cost;
    for (int i = 0; i < idx; i += 2) {
        f[i] += f[i ^ 1], f[i ^ 1] = 0;
        w[i] = -w[i], w[i ^ 1] = -w[i ^ 1];
    }
    EK(flow, cost);
    ansmax = -cost;
    cout << ansmin << '\n';
    cout << ansmax << '\n';
}
```

多源汇最大流

建超级源点和汇点，所有超级源点向所有源点连边，所有汇点向超级汇点连边

```
int n, m, S, T;
int h[N], e[M], f[M], ne[M], idx;
int d[N], cur[N];
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    int sc, tc;
    cin >> n >> m >> sc >> tc;
    S = 0, T = n + 1;
    memset(h, -1, sizeof(h));
    while (sc--) {
        int x;
        cin >> x;
        add(S, x, INF);
    }
    while (tc--) {
        int x;
        cin >> x;
        add(x, T, INF);
    }
```

```
    while (m--) {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c);
    }
    cout << dinic() << '\n';
}
```

多源汇费用流

```
int n, m, S, T;
int h[N], e[M], f[M], w[M], ne[M], idx;
int d[N], pre[N], incf[N];
bool st[N];
void add(int a, int b, int c, int d) {
    e[idx] = b, f[idx] = c, w[idx] = d, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, w[idx] = -d, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    cin >> m >> n;
    S = 0, T = m + n + 1;
    memset(h, -1, sizeof(h));
    for (int i = 1; i <= m; i++) {
        int a;
        cin >> a;
        add(S, i, a, 0);
    }
    for (int i = 1; i <= n; i++) {
        int a;
        cin >> a;
        add(m + i, T, a, 0);
    }
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            int c;
            cin >> c;
            add(i, m + j, INF, c);
        }
    }
    int flow, cost;
    EK(flow, cost);
    cout << cost << '\n'; // 最小费用

    for (int i = 0; i < idx; i += 2) {
        f[i] += f[i ^ 1];
        f[i ^ 1] = 0;
        w[i] = -w[i], w[i ^ 1] = -w[i ^ 1];
    }
    EK(flow, cost);
    cout << -cost << '\n'; // 最大费用
}
```

上下界可行流

无源汇上下界可行流

```
int n, m, S, T;
int h[N], e[M], f[M], l[M], ne[M], idx;
int d[N], cur[N], A[N];
void add(int a, int b, int c, int d) { // c:下界，d:上界
    e[idx] = b, f[idx] = d - c, l[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    cin >> n >> m;
    S = 0, T = n + 1;
    memset(h, -1, sizeof(h));
    for (int i = 0; i < m; i++) {
        int a, b, c, d;
        cin >> a >> b >> c >> d;
        add(a, b, c, d);
        A[a] -= c, A[b] += c;
    }
    int tot = 0;
    for (int i = 1; i <= n; i++) {
        if (A[i] > 0) {
            add(S, i, 0, A[i]);
            tot += A[i];
        }
```

```
        else if (A[i] < 0) add(i, T, 0, -A[i]);
    }
    if (dinic() != tot) cout << "NO" << '\n';
    else {
        cout << "YES" << '\n';
        for (int i = 0; i < m * 2; i += 2) cout << f[i ^ 1] + l[i] << '\n';
    }
}
```

有源汇上下界最大流

```
int n, m, S, T;
int h[N], e[M], f[M], ne[M], idx;
int d[N], cur[N], A[N];
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    int s, t;
    cin >> n >> m >> s >> t;
    S = 0, T = n + 1;
    memset(h, -1, sizeof(h));
    for (int i = 0; i < m; i++) {
        int a, b, c, d;
        cin >> a >> b >> c >> d;
        add(a, b, d - c);
        A[a] -= c, A[b] += c;
    }
    int tot = 0;
    for (int i = 1; i <= n; i++) {
        if (A[i] > 0) {
            add(S, i, A[i]);
            tot += A[i];
        }
        else if (A[i] < 0) add(i, T, -A[i]);
    }
    add(t, s, INF);
    if (dinic() < tot) cout << "No Solution" << '\n';
    else {
        int res = f[idx - 1];
        S = s, T = t;
        f[idx - 1] = f[idx - 2] = 0;
        cout << res + dinic() << '\n';
    }
}
```

有源汇上下界最小流

```
int n, m, S, T;
int h[N], e[M], f[M], ne[M], idx;
int d[N], cur[N], A[N];
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void solve() {
    int s, t;
    cin >> n >> m >> s >> t;
    S = 0, T = n + 1;
    memset(h, -1, sizeof(h));
    for (int i = 0; i < m; i++) {
        int a, b, c, d;
        cin >> a >> b >> c >> d;
        add(a, b, d - c);
        A[a] -= c, A[b] += c;
    }
    int tot = 0;
    for (int i = 1; i <= n; i++) {
        if (A[i] > 0) {
            add(S, i, A[i]);
            tot += A[i];
        }
        else if (A[i] < 0) add(i, T, -A[i]);
    }
    add(t, s, INF);
    if (dinic() < tot) cout << "No Solution" << '\n';
    else {
        int res = f[idx - 1];
        S = t, T = s;
```

```
        f[idx - 1] = f[idx - 2] = 0;
        cout << res - dinic() << '\n';
    }
}
```

最大流关键边

求关键边数量

```
int n, m, S, T;
int h[N], e[M], f[M], ne[M], idx;
int d[N], cur[N];
bool vis_s[N], vis_t[N];
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void dfs(int u, bool st[], int t) {
    st[u] = 1;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = i ^ t;
        int ver = e[i];
        if (f[j] && !st[ver]) dfs(ver, st, t);
    }
}
void solve() {
    cin >> n >> m;
    S = 0, T = n - 1;
    memset(h, -1, sizeof(h));
    for (int i = 0; i < m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c);
    }
    dinic();
    dfs(S, vis_s, 0);
    dfs(T, vis_t, 1);
    int res = 0;
    for (int i = 0; i < m * 2; i += 2) {
        if (!f[i] && vis_s[e[i ^ 1]] && vis_t[e[i]]) {
            res++;
        }
    }
    cout << res << '\n';
}
```

最大密度子图

边的数量 (E + V * 2) * 2

无点权无边权

```
int n, m, S, T;
int h[N], e[M], ne[M], idx;
double f[M];
int d[N], cur[N];
int dg[N];
struct Edge {
    int a, b;
}edges[M];
int ans;
bool st[N];
void add(int a, int b, double c1, double c2) {
    e[idx] = b, f[idx] = c1, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = c2, ne[idx] = h[b], h[b] = idx++;
}
bool bfs() {
    queue<int> q;
    memset(d, -1, sizeof(d));
    q.push(S), d[S] = 0, cur[S] = h[S];
    while (q.size()) {
        auto t = q.front();
        q.pop();
        for (int i = h[t]; i != -1; i = ne[i]) {
            int ver = e[i];
            if (d[ver] == -1 && f[i] > 0) {
                d[ver] = d[t] + 1;
                cur[ver] = h[ver];
                q.push(ver);
```

```
                if (ver == T) return 1;
            }
        }
    }
    return 0;
}
double find(int u, double limit) {
    if (u == T) return limit;
    double flow = 0;
    for (int i = cur[u]; i != -1 && flow < limit; i = ne[i]) {
        cur[u] = i;
        int ver = e[i];
        if (d[ver] == d[u] + 1 && f[i] > 0) {
            double t = find(ver, min(f[i], limit - flow));
            if (t < eps) d[ver] = -1;
            f[i] -= t, f[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}
void build(double g) {
    memset(h, -1, sizeof(h));
    idx = 0;
    for (int i = 0; i < m; i++) add(edges[i].a, edges[i].b, 1, 1);
    for (int i = 1; i <= n; i++) {
        add(S, i, m, 0);
        add(i, T, m + 2 * g - dg[i], 0);
    }
}
double dinic(double g) {
    build(g);
    double r = 0, flow;
    while (bfs()) while (flow = find(S, INF)) r += flow;
    return r;
}
void dfs(int u) {
    st[u] = 1;
    if (u != S) ans++;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int ver = e[i];
        if (!st[ver] && f[i] > 0) dfs(ver);
    }
}
void solve() {
    cin >> n >> m;
    S = 0, T = n + 1;
    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        dg[a]++, dg[b]++;
        edges[i] = {a, b};
    }
    double l = 0, r = m;
    while (r - l > eps) {
        double mid = (l + r) / 2;
        double t = dinic(mid);
        if (m * n - t > 0) l = mid;
        else r = mid;
    }
    dinic(l);
    dfs(S);
    if (!ans) {
        cout << 1 << '\n';
        cout << 1 << '\n';
        return;
    }
    cout << ans << '\n';
    for (int i = 1; i <= n; i++) {
        if (st[i]) cout << i << '\n';
    }
}
```

最大权闭合子图

最大权为 正权和 - 最小割

源点向正权点连权值边，负权点向汇点连权值的绝对值，有向图原来的边的权值为无限大

```
int n, m, S, T;
int h[N], e[M], f[M], ne[M], idx;
```

```
int d[N], cur[N];
void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
void dfs(int u) {
    st[u] = 1;
    for (int i = h[u]; i != -1; i = ne[i]) {
        if (!st[e[i]] && f[i]) {
            dfs(e[i]);
        }
    }
}
void solve() {
    cin >> n >> m;
    S = 0, T = n + m + 1;
    memset(h, -1, sizeof(h));
    for (int i = 1; i <= n; i++) {
        int p;
        cin >> p;
        add(m + i, T, p);
    }
    int tot = 0;
    for (int i = 1; i <= m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        add(S, i, c);
        add(i, m + a, INF);
        add(i, m + b, INF);
        tot += c;
    }
    cout << tot - dinic() << '\n';
    dfs(S); // 求方案
    for (int i = 1; i <= m; i++) {
        if (st[i]) cout << i << ' ';
    }
    cout << '\n';
    for (int i = m + 1; i <= m + n; i++) {
        if (st[i]) cout << i - m << ' ';
    }
}
```

字符串

本质不同的子序列

```
// dp[i] 强制以 s[i] 为结尾得本质不同子序列个数
vector<long long> pre(n + 1, 0), dp(n + 1, 0);
vector<int> last(26, 0);
pre[0] = 1;
for (int i = 1; i <= n; i++) {
    int ch = s[i] - 'a';
    if (!last[ch]) dp[i] = pre[i - 1];
    else dp[i] = (pre[i - 1] - pre[last[ch] - 1]) % mod;
    pre[i] = (pre[i - 1] + dp[i]) % mod;
    last[ch] = i;
}
int ans = ((pre[n] - 1) % mod + mod) % mod;
```

```
// dp[i] 为前 i 个字母构成的所有本质不同子序列个数
vector<int> last(26, -1);
for (int i = 1; i <= n; i++) {
    dp[i] = (dp[i - 1] * 2 + 1) % mod;
    int ch = s[i] - 'a';
    if (last[ch] >= 0) dp[i] -= (dp[last[ch] - 1] + 1);
    dp[i] = (dp[i] % mod + mod) % mod;
    last[ch] = i;
}
```

回文自动机

```
namespace pam { // 当需要跳多次时可以使用 ori, 和 ccnt 来记录上次状态
    int sz, tot, last;
    int cnt[N], ch[N][26], len[N], fail[N]; // cnt 是以每个节点结尾的回文子串个数, len 是每个回文子串的长度
    // int ori[N], ccnt[N];
    char s[N];
    int node(int l) {  // 建立一个新节点, 长度为 l
        sz++;
```

```
        memset(ch[sz], 0, sizeof(ch[sz]));
        len[sz] = l;
        fail[sz] = cnt[sz] = 0;
        return sz;
    }
    void clear() {  // 初始化
        sz = -1;
        last = 0;
        s[tot = 0] = '$';
        node(0);
        node(-1);
        fail[0] = 1;
    }
    int getfail(int x) {  // 找后缀回文
        while (s[tot - len[x] - 1] != s[tot]) x = fail[x];
        return x;
    }
    void insert(char c) {  // 建树
        s[++tot] = c;
        int now = getfail(last);
        if (!ch[now][c - 'a']) {
            int x = node(len[now] + 2);
            fail[x] = ch[getfail(fail[now])][c - 'a'];
            ch[now][c - 'a'] = x;
            // cnt[x] = cnt[fail[x]] + 1; 可以在建立的时候直接算
        }
        last = ch[now][c - 'a'];
        cnt[last]++; // 如果在建立的时候算需要去掉
    }
    long long solveans(bool ok, int n) { // ok = 1恢复， ok == 0正常
        long long ans = 0;
        // for (int i = 0; i <= sz; i++) ori[i] = cnt[i];
        for (int i = sz; i >= 0; i--) cnt[fail[i]] += cnt[i];
        for (int i = 1; i <= sz; i++) {
            /*
            if (ok) {
                ccnt[i] = cnt[i];
                continue;
            }
            */
            for (int i = 1; i <= sz; i++) ans = max(ans, 1ll * len[i] * cnt[i]);
        }
        /*
        if (ok) {
            for (int i = 0; i <= sz; i++) {
                cnt[i] = ori[i];
            }
        }
        */
        return ans;
    }
}
void solve() {
    int n;
    cin >> n;
    string s;
    cin >> s;
    pam::clear();
    for (auto i : s) pam::insert(i);
    cout << pam::solveans(0, s.size()) << "\n";
}
```

字符串哈希

核心思想：将字符串看成 P 进制数，P 的经验值是 131 或 13331，取这两个值的冲突概率低，自然溢出容易被卡，应取模大数

```
const int P = 131, PP = 1331;
i128 MOD = 21237044013013795711;
i128 h1[N], p1[N], h2[N], p2[N]; // h[k] 存储字符串前 k 个字母的哈希值
// 初始化
p1[0] = 1, p2[0] = 1;
for (int i = 1; i <= n; i++) {
    h1[i] = ((h1[i - 1] * P + str[i]) % MOD + MOD) % MOD;
    p1[i] = (p1[i - 1] * P % MOD + MOD) % MOD;
    h2[i] = ((h2[i - 1] * PP + str[i]) % MOD + MOD) % MOD;
    p2[i] = (p2[i - 1] * PP % MOD + MOD) % MOD;
}
// 计算子串 str[l ~ r] 的哈希值
i128 get(i128 h[], i128 p[], int l, int r) {
    i128 ans = ((h[r] - h[l - 1] * p[r - l + 1] % MOD) % MOD + MOD) % MOD;
```

```
    return ans;
}
```

最大回文子串

可以字符串哈希

manacher 算法

O(n)

```
int p[N];
string x;
string t;
void init() {
    t += "$#";
    for (int i = 0; i < x.size(); i++) {
        t += x[i];
        t += '#';
    }
    t += '^';
}
int ans = 1;
void manacher() {
    int mr = 0, mid = 0;
    for (int i = 1; i < t.size(); i++) {
        if (i < mr) p[i] = min(p[2 * mid - i], mr - i);
        else p[i] = 1;
        while (t[i - p[i]] == t[i + p[i]]) p[i]++;
        if (i + p[i] > mr) {
            mr = i + p[i];
            mid = i;
        }
        if (i % 2 == 0) ans = max(ans, p[i] / 2 + p[i] / 2 - 1);
        else ans = max(ans, p[i] - 1);
    }
}
```

最小表示法

判断两字符串最小表示法是否相同

```
int get_min(string s, int len) {
    int i = 0, j = 1;
    while (i < len && j < len) {
        int k = 0;
        while (k < len && s[i + k] == s[j + k]) k++;
        if (k == len) break;
        if (s[i + k] > s[j + k]) i += k + 1;
        else j += k + 1;
        if (i == j) j++;
    }
    int k = min(i, j);
    return k;
}
void solve() {
    string a, b;
    cin >> a >> b;
    int n = a.size(), m = b.size();
    a += a, b += b;
    int x = get_min(a, n), y = get_min(b, m);
    if (a.substr(x, n) != b.substr(y, m)) cout << "No" << '\n';
    else {
        cout << "Yes" << '\n';
        cout << a.substr(x, n) << '\n';
    }
}
```

AC自动机

n 个模式串和一个文本串, 求有多少个不同的模式串在文本串里出现过

```
int n;
int tr[N][26], cnt[N], idx;
int fail[N];
bool vis[N];
void insert(string str) {
    int p = 0;
    for (int i = 0; i < str.size(); i++) {
```

```
        int t = str[i] - 'a';
        if (!tr[p][t]) tr[p][t] = ++idx;
        p = tr[p][t];
    }
    cnt[p]++;
}
void build() {
    queue<int> q;
    for (int i = 0; i < 26; i++) {
        if (tr[0][i]) q.push(tr[0][i]);
    }
    while (q.size()) {
        int t = q.front();
        q.pop();
        for (int i = 0; i < 26; i++) {
            int p = tr[t][i];
            if (!p) tr[t][i] = tr[fail[t]][i];
            else {
                fail[p] = tr[fail[t]][i];
                q.push(p);
            }
        }
    }
}
void solve() {
    cin >> n;
    for (int i = 1; i <= n; i++) {
        string str;
        cin >> str;
        insert(str);
    }
    build();
    string str;
    cin >> str;
    int res = 0;
    for (int i = 0, j = 0; i < str.size(); i++) {
        int t = str[i] - 'a';
        j = tr[j][t];
        int p = j;
        while (!vis[p] && p) {
            res += cnt[p];
            cnt[p] = 0;
            vis[p] = 1;
            p = fail[p];
        }
    }
    // for (int i = 1; i <= idx; i++) g[fail[i]].push_back(i); 可以建立fail树在树上统计信息
    cout << res << '\n';
}
```

fail 树上 dp

```
int tr[N][26], idx, id[300];
int fail[N];
i64 f[N][N][256 + 5];
void insert(string str, int sid) {
    int p = 0;
    for (int i = 0; i < str.size(); i++) {
        int t = str[i] - 'a';
        if (!tr[p][t]) tr[p][t] = ++idx;
        p = tr[p][t];
    }
    id[p] |= 1 << sid;
}
void build() {
    queue<int> q;
    for (int i = 0; i < 26; i++) {
        if (tr[0][i]) q.push(tr[0][i]);
    }
    while (q.size()) {
        int t = q.front();
        q.pop();
        id[t] |= id[fail[t]];
        for (int i = 0; i < 26; i++) {
            int p = tr[t][i];
            if (!p) tr[t][i] = tr[fail[t]][i];
            else {
                fail[p] = tr[fail[t]][i];
                q.push(p);
            }
```

```
        }
    }
}
void solve() {
    int n, L;
    cin >> n >> L;
    for (int i = 0; i < n; i++) {
        string x;
        cin >> x;
        insert(x, i);
    }
    build();
    f[0][0][0] = 1;
    for (int i = 0; i < L; i++) {
        for (int j = 0; j <= idx; j++) {
            for (int S = 0; S < 1 << n; S++) {
                if (!f[i][j][S]) continue;
                for (int c = 0; c < 26; c++) {
                    int to = tr[j][c];
                    f[i + 1][to][S | id[to]] += f[i][j][S];
                    f[i + 1][to][S | id[to]] %= mod;
                }
            }
        }
    }
    i64 ans = 0;
    for (int i = 0; i <= idx; i++) ans = (ans + f[L][i][(1 << n) - 1]) % mod;
    cout << ans << '\n';
}
```

KMP

字符串的最小循环长度 = m - ne[m]

p 是周期，len(s) - p 是 border

s 是长文本，p 是模式串，n 是 s 的长度，m 是 p 的长度

求模式串的 Next 数组：ne[i] 就是模式串前缀 i 的 border

```
for (int i = 2, j = 0; i <= m; i++) {
    while (j && p[i] != p[j + 1]) j = ne[j];
    if (p[i] == p[j + 1]) j++;
    ne[i] = j;
}
```

匹配

```
for (int i = 1, j = 0; i <= n; i++) {
    while (j && s[i] != p[j + 1]) j = ne[j];
    if (s[i] == p[j + 1]) j++;
    if (j == m) {
        j = ne[j];
        // 匹配成功后的逻辑
        // cout << i - m + 1 << '\n'; 模式串在文本串中的开头
    }
}
```

SA

sa[i]:排名第 i 位的是第几个后缀，rk[i]：第 i 个后缀排第几，height[i]：sa[i] 和 sa[i - 1]的最长公共前缀 lcp(i - 1, i)

lcp(i, j) = min(lcp(i, k), lcp(k, j)) i <= k <= j lcp(i, j) = min(lcp(i, i + 1), lcp(i + 1, ...), lcp(j - 1, j))

```
const int N = 1000010;
int n, m;
string s;
int sa[N], x[N], y[N], c[N], rk[N], height[N];
void get_sa() {
    for (int i = 1; i <= n; i++) c[x[i] = s[i]]++;
    for (int i = 2; i <= m; i++) c[i] += c[i - 1];
    for (int i = n; i; i--) sa[c[x[i]]--] = i;
    for (int k = 1; k <= n; k <<= 1) {
        int num = 0;
        for (int i = n - k + 1; i <= n; i++) y[++num] = i;
        for (int i = 1; i <= n; i++) {
            if (sa[i] > k) {
                y[ ++ num] = sa[i] - k;
```

```
            }
        }
        for (int i = 1; i <= m; i++) c[i] = 0;
        for (int i = 1; i <= n; i++) c[x[i]]++ ;
        for (int i = 2; i <= m; i++) c[i] += c[i - 1];
        for (int i = n; i; i--) sa[c[x[y[i]]]--] = y[i], y[i] = 0;
        swap(x, y);
        x[sa[1]] = 1, num = 1;
        for (int i = 2; i <= n; i++) x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + k] == y[sa[i - 1] + k]) ? num :
++num;
        if (num == n) break;
        m = num;
    }
}
void get_height() {
    for (int i = 1; i <= n; i++) rk[sa[i]] = i;
    for (int i = 1, k = 0; i <= n; i++) {
        if (rk[i] == 1) continue;
        if (k) k--;
        int j = sa[rk[i] - 1];
        while (i + k <= n && j + k <= n && s[i + k] == s[j + k]) k++;
        height[rk[i]] = k;
    }
}
void solve() {
    cin >> s;
    n = s.size(), m = 122;
    s = ' ' + s;
    get_sa();
    get_height();
    for (int i = 1; i <= n; i++) cout << sa[i] << ' ';
    cout << '\n';
    for (int i = 1; i <= n; i++) cout << height[i] << ' ';
}
```

本质不同的子串

所有后缀的前缀集合就是所有子串集合

ans = i from 1 - n (len[rk[i]] - height[rk[i]]) = n * (n + 1) / 2 - sum(height)

SAM

SAM 是个状态机，原串的所有子串和从 SAM 起点开始的所有路径一一对应，所以终点就是包含后缀的点。

每个点包含若干子串，每个子串都一一对应一条从起点到该点的路径。且这些子串一定是里面最长子串的连续后缀。

一个状态表示的是出现位置相同，但长度不同的一系列串

endpos(s)：子串 s 所有出现的位置（尾字母下标）集合。SAM 中的每个状态都一一对应一个 endpos 的等价类。

令 s1, s2 为 S 的两个子串，不妨设 |s1| ≤ |s2|。则 s1 是 s2 的后缀当且仅当 endpos(s1) ⊇ endpos(s2)，s1 不是 s2 的后缀当且仅当 endpos(s1) ∩ endpos(s2) = ∅

两个子串的 endpos 相同，那么短串为长串的后缀。

对于一个状态 st，以及任意的 longest(st) 的后缀 s，如果 s 的长度满足：|shortest(st)| ≤ |s| ≤ |longsest(st)|，那么 s ∈ substrings(st)。

本质不同子串数量

求和 SAM 上每个点表示的子串数量

出现次数不为 1 的子串出现次数 * 子串长度的最大值

每个子串出现的次数就是对应 endpos 的大小

node[u].len 是状态 u 所能表示的最长子串的长度

f[u] 以状态 u 表示的所有子串，在整个原串中出现了多少次

```
const int N = 2e6 + 10;
int tot = 1, last = 1;
struct Node {
    int len, fa;
    int ch[26];
} node[N];
string str;
LL f[N], ans;
vector<vector<int>> g(N);
```

```
void extend(int c) {
    int p = last, np = last = ++tot;
    f[tot] = 1;
    node[np].len = node[p].len + 1;
    for (; p && !node[p].ch[c]; p = node[p].fa) node[p].ch[c] = np;
    if (!p) node[np].fa = 1;
    else {
        int q = node[p].ch[c];
        if (node[q].len == node[p].len + 1) node[np].fa = q;
        else {
            int nq = ++tot;
            node[nq] = node[q], node[nq].len = node[p].len + 1;
            node[q].fa = node[np].fa = nq;
            for (; p && node[p].ch[c] == q; p = node[p].fa) node[p].ch[c] = nq;
        }
    }
}
void dfs(int u) {
    for (auto v : g[u]) {
        dfs(v);
        f[u] += f[v];
    }
    if (f[u] > 1) ans = max(ans, f[u] * node[u].len);
}
void solve() {
    cin >> str;
    for (auto u : str) extend(u - 'a');
    for (int i = 2; i <= tot; i++) g[node[i].fa].push_back(i);
    dfs(1);
    cout << ans << '\n';
}
```

求匹配串的（最长的）前缀是模式串上的子串的长度

只要在 SAM 上暴力走就行

n 个字符串的最长公共子串

```
const int N = 20010;
int n;
int tot = 1, last = 1;
string str;
struct Node {
    int len, fa;
    int ch[26];
} node[N];
int ans[N], now[N];
vector<vector<int>> g(N);
void extend(int c) {
    int p = last, np = last = ++tot;
    node[np].len = node[p].len + 1;
    for (; p && !node[p].ch[c]; p = node[p].fa) node[p].ch[c] = np;
    if (!p) node[np].fa = 1;
    else {
        int q = node[p].ch[c];
        if (node[q].len == node[p].len + 1) node[np].fa = q;
        else {
            int nq = ++tot;
            node[nq] = node[q], node[nq].len = node[p].len + 1;
            node[q].fa = node[np].fa = nq;
            for (; p && node[p].ch[c] == q; p = node[p].fa) node[p].ch[c] = nq;
        }
    }
}
void dfs(int u) {
    for (auto v : g[u]) {
        dfs(v);
        now[u] = max(now[u], now[v]);
    }
}
void solve() {
    cin >> n;
    cin >> str;
    for (auto u : str) extend(u - 'a');
    for (int i = 1; i <= tot; i++) ans[i] = node[i].len;
    for (int i = 2; i <= tot; i++) g[node[i].fa].push_back(i);

    for (int i = 0; i < n - 1; i++) {
        cin >> str;
        memset(now, 0, sizeof(now));
        int p = 1, t = 0;
```

```
        for (auto u : str) {
            int c = u - 'a';
            while (p > 1 && !node[p].ch[c]) p = node[p].fa, t = node[p].len;
            if (node[p].ch[c]) p = node[p].ch[c], t++;
            now[p] = max(now[p], t);
        }
        dfs(1);
        for (int j = 1; j <= tot; j++) ans[j] = min(ans[j], now[j]);
    }
    int res = 0;
    for (int i = 1; i <= tot; i++) res = max(res, ans[i]);
    cout << res << '\n';
}
```

杂项

蔡勒

```
int get(int year, int month, int day) {
    if (month <= 2) {
        month += 12;
        year--;
    }
    int c = year / 100, y = year % 100, m = month, d = day;
    int w = y + y / 4 + c / 4 - 2 * c + 26 * (m + 1) / 10 + d - 1;
    w = (w % 7 + 7) % 7;
    return w;
}
```

对顶堆求动态中位数

```
void solve() {
    int n;
    cin >> n;
    priority_queue<int> down;
    priority_queue<int, vector<int>, greater<int>> up;
    for (int i = 1; i <= n; i++) {
        int x;
        cin >> x;
        if (down.empty() || x <= down.top()) down.push(x);
        else up.push(x);
        if (down.size() > up.size() + 1) {
            up.push(down.top());
            down.pop();
        }
        if (up.size() > down.size()) {
            down.push(up.top());
            up.pop();
        }
        if (i % 2) cout << down.top() << ' ';
    }
}
```

对拍

```
import subprocess
import difflib
import os

DATA_CPP = "data.cpp"
BF_CPP = "bf.cpp"
MAIN_CPP = "main.cpp"

DATA_EXE = "./data"
BF_EXE = "./bf"
MAIN_EXE = "./main"

# 判断平台自动添加.exe
if os.name == 'nt':
    DATA_EXE = "data.exe"
    BF_EXE = "bf.exe"
    MAIN_EXE = "main.exe"

# 编译阶段
print("正在编译...")
subprocess.run(["g++", DATA_CPP, "-o", DATA_EXE], check=True)
subprocess.run(["g++", BF_CPP, "-o", BF_EXE], check=True)
subprocess.run(["g++", MAIN_CPP, "-o", MAIN_EXE], check=True)
```

```
print("开始对拍...")

for i in range(1, 1001):
    # 生成数据
    with open("input.txt", "w") as f:
        subprocess.run([DATA_EXE], stdout=f)

    # 暴力输出
    with open("input.txt", "r") as fin, open("ans_bf.txt", "w") as fout:
        subprocess.run([BF_EXE], stdin=fin, stdout=fout)

    # 优化输出
    with open("input.txt", "r") as fin, open("ans_main.txt", "w") as fout:
        subprocess.run([MAIN_EXE], stdin=fin, stdout=fout)

    # 读取两个结果
    with open("ans_bf.txt") as f1, open("ans_main.txt") as f2:
        ans1 = f1.readlines()
        ans2 = f2.readlines()

    if ans1 != ans2:
        print(f"第 {i} 组数据输出不一致！")
        print("输入数据：")
        with open("input.txt") as fin:
            print(fin.read())
        print("正确答案：")
        print("".join(ans1))
        print("你的输出：")
        print("".join(ans2))
        print("差异：")
        diff = difflib.unified_diff(ans1, ans2, fromfile='bf', tofile='main')
        print("".join(diff))
        break
    else:
        print(f"第 {i} 组通过")
```

根号调整

```
i128 sqrtup(i128 x) { // 返回大于等于根号的第一个数
    i128 d = sqrtl(x);
    while (d * d > x) d--;
    while (d * d < x) d++;
    return d;
}
i128 sqrtdn(i128 x) { // 返回小于等于根号的第一个数
    i128 d = sqrtl(x);
    while (d * d < x) d++;
    while (d * d > x) d--;
    return d;
}
```

环形均分

```
int a[N];
LL c[N], s[N];
LL work(int n, int a[]) {
    for (int i = 1; i <= n; i++) s[i] = s[i - 1] + a[i];
    if (s[n] % n) return -1;
    LL avg = s[n] / n;
    c[1] = 0;
    for (int i = 2; i <= n; i++) c[i] = s[i - 1] - (i - 1) * avg;
    sort(c + 1, c + n + 1);
    LL res = 0;
    for (int i = 1; i <= n; i++) res += abs(c[i] - c[n + 1 >> 1]);
    return res;
}
```

康托展开

```
#include<bits/stdc++.h>
using namespace std;
#define MAXN 1000005
#define mod 998244353
int n, a[MAXN], fac, c[MAXN], ans;
char *p;
inline void read(int &x) {
```

```
  x = 0;
    while (!isdigit(*p)) ++p;
  while (isdigit(*p) ) x = x * 10 + (*p & 15), ++p;
}
int main() {
  cin >> n;
    fac = 1;
  p = new char[n * 8 + 100],
  fread(p, 1, n * 8 + 100, stdin);
  for (int i = n; i; --i) read(a[i]);
  for (int i = 1, s, j; i <= n; ++i) {
    for (s = 0, j = a[i]; j; j -= j & -j) s += c[j];
    ans = (ans + 1ll * fac * s) % mod, fac = 1ll * fac * i % mod;
    for (j = a[i]; j <= n; j += j & -j) ++c[j];
  }
    cout << ans + 1 << '\n';
  return 0;
}
```

逆序对

```
int tmp[N];
LL cnt;
void merge_sort(int q[], int l, int r) {
    if (l >= r) return;
    int mid = (l + r) >> 1;
    merge_sort(q, l, mid);
    merge_sort(q, mid + 1, r);
    int i = l, j = mid + 1, k = 0;
    while (i <= mid && j <= r) {
        if (q[i] <= q[j]) tmp[k++] = q[i++];
        else {
            tmp[k++] = q[j++];
            cnt += (mid - i + 1);
        }
    }
    while (i <= mid) tmp[k++] = q[i++];
    while (j <= r) tmp[k++] = q[j++];
    for (i = l, j = 0; i <= r; i++, j++) q[i] = tmp[j];
}
```

三维差分

```
D[x1][y1][z1]       += d;   //前面：左下顶点，即区间的起始点
D[x2+1][y1][z1]     -= d;   //前面：右下顶点的右边一个点
D[x1][y1][z2+1]     -= d;   //前面：左上顶点的上面一个点
D[x2+1][y1][z2+1]   += d;   //前面：右上顶点的斜右上方一个点
D[x1][y2+1][z1]     -= d;   //后面：左下顶点的后面一个点
D[x2+1][y2+1][z1]   += d;   //后面：右下顶点的斜右后方一个点
D[x1][y2+1][z2+1]   += d;   //后面：左上顶点的斜后上方一个点
D[x2+1][y2+1][z2+1] -= d;   //后面：右上顶点的斜右上后方一个点，即区间终点的后一个点
```

输出txt

```
ofstream outfile("output.txt");
void solve() {
    vector<int> a(4);
    a[0] = 0, a[1] = 1, a[2] = 2, a[3] = 3;
    for (int i = 0; i < 4; i++) outfile << i << ' ' << a[i] << '\n';
}
```

跳跃游戏

```
int jump(vector<int>& nums) {
    int maxPos = 0, n = nums.size(), end = 0, step = 0;
    for (int i = 0; i < n - 1; i++) {
        if (maxPos >= i) {
            maxPos = max(maxPos, i + nums[i]);
            if (i == end) {
                end = maxPos;
                ++step;
            }
        }
    }
    return step;
}
```

约瑟夫环加强

```
int a[N];
void solve() {
    /*
    普通约瑟夫环
    int n, k;
    cin >> n >> k;
    for (int i = 1; i <= n; i++) a[i] = i;
    */
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    int ans = 1;
    for (int c = 2, i = n - 1; c <= n; c++, i--) ans = (ans + a[i] - 1) % c + 1;
    cout << ans << '\n';
}
```

INT_128

```
void put(i128 x) {
    vector<int> s;
    do {
        s.push_back(x % 10);
        x /= 10;
    } while (x);
    while (!s.empty()) {
        cout << s.back();
        s.pop_back();
    }
    cout << "\n";
}
```

初始化

```
#include<bits/stdc++.h>
using namespace std;
mt19937_64 rnd(chrono::steady_clock::now().time_since_epoch().count());
using i64 = long long;
using u64 = unsigned long long;
using i128 = __int128;

void solve() {

}

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    int T = 1;
    cin >> T;
    while (T--) solve();
    return 0;
}
```