ACM常用模板-morejarphone2.0

ACM常用模板-morejarphone2.0
- 几何
  - 平面几何
    - 一些公式
    - 常用模板
  - 扫描线
  - 三维几何
  - 平面最近点对
  - 杂题
- 图论
  - 一些结论
  - 图上三元环计数
  - 最大团
  - 生成树
    - 最小生成树
    - 最小基环森林
    - 生成树计数(matrix-tree定理)
    - 最优比率生成树
    - 最小树形图
  - 最短路&&差分约束
    - dijstra
    - SPFA
    - floyd
    - 最小字典序单源最短路
    - 最优比率环
    - 差分约束
    - 同余最短路
  - 平面图
    - 对偶图
    - 一类平面图上最小割
  - 连通性及缩点
    - 求割点
    - 求桥
    - 点双联通分量
    - 边双联通分量
    - 有向图强连通分量
  - 拓扑排序
  - 2-sat
  - 二分图
    - 二分图的一些结论
    - 二分图最大权匹配
    - 最小路径覆盖
  - 网络流
    - 最大流&&最小割
    - 费用流

- 有上下界网络流
- 其他图论
    - 弦图
    - 欧拉路径
    - prufer序列
- 数学
    - 基本数论&&推导
        - 一些结论
        - 求最大的模
        - 求(0^n)+(1^n)+...+((n-1)^n)
        - 求B进制下n以内的回文数字
        - 快速乘法
        - 快速幂
        - 逆元
        - [1,n]内素数个数
        - 微积分
        - 拉格朗日插值
    - 多项式
    - 扩展欧几里得
    - 大素数测试和大数分解
    - pell方程
    - 线性递推
    - 二次剩余
    - 博弈
        - 威佐夫博弈
        - NIM博弈
            - sg函数
            - nim积
        - 必败态扩充
        - 树上删边博弈
    - 矩阵
        - 矩阵快速幂
    - 线性筛(质数 欧拉函数 最小素因子 莫比乌斯函数)
    - 欧拉函数
        - 欧拉公式降幂
        - 求解欧拉函数
    - 积性函数求和
    - 组合数学(容斥&&反演等)
        - 一些结论
        - 枚举子集
        - 第二类斯特林数
        - 组合计数
        - 容斥原理
        - 莫比乌斯反演
        - 分块加速
        - Lucas定理
    - 二分&&迭代&&三分
    - 高斯消元
    - 线性基
        - XOR线性基
    - 卡特兰数

- 模方程相关
  - 多元线性同余方程
  - 中国剩余定理
- FFT&&(NTT)
  - 原根
  - 关于卷积的FFT
  - 任意模数的FFT
  - 关于计数的FFT
  - NTT
- FWT

一般数据结构
- STL相关
  - 哈希表
  - 单调栈
- 并查集
  - 种类并查集
- RMQ
- 树状数组
  - 子矩阵更新单点求和
  - 区间中出现的数字种数
- 线段树
- 树链剖分
- 最近公共祖先
- 笛卡尔树
- 平衡树
  - treap树
  - splay树
- 树分治
  - 一类树路径问题
  - 点分治序列
- 虚树
- 左偏树
- KD树
- 莫队
- 主席树

字符串相关数据结构
- 哈希
- KMP
- 最大(小)表示法
- manachar
- 字典树
  - 01字典树
- AC自动机
- 后缀数组
- 基本子串字典(HDU 5814)

动态规划
- 普通DP
  - 有趣的转移
  - 字典树DP
  - AC自动机DP
  - LIS

- 数位DP
- 概率(期望)DP
- 斜率优化
- 背包

搜索
- dfs
  - 剪枝
  - 随机背包(大背包)

其他
- 贪心
- 构造
- 前后缀
  - 最大子序列/矩阵和
- 分治&&CDQ分治
  - 三维偏序

拓展
- java
- 输入输出外挂
- 求某天是星期几
- 扩栈
- 头文件组
- vimrc简单配置

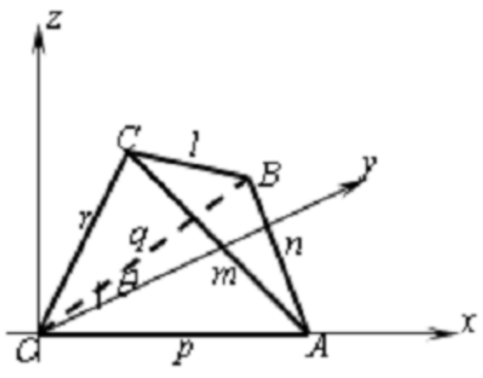
几何

平面几何

一些公式

- 欧拉四面体公式：

  1，建议x，y，z直角坐标系。设A、B、C少拿点的坐标分别为(a1,b,1,c1),(a2,b2,c2),(a3,b3,c3),四面体O-ABC的六条棱长分别为l，m，n，p，q，r；



$$V^2 = \frac{1}{36}\begin{vmatrix} p^2 & \frac{p^2+q^2-n^2}{2} & \frac{p^2+r^2-m^2}{2} \\ \frac{p^2+q^2-n^2}{2} & q^2 & \frac{q^2+r^2-l^2}{2} \\ \frac{p^2+r^2-m^2}{2} & \frac{q^2+r^2-l^2}{2} & r^2 \end{vmatrix}$$

- 海伦公式： 三角形面积版本：$S = \sqrt{p(p-a)(p-b)(p-c)}$,其中$p$是半周长 圆内接四边形版本：$S = \sqrt{(p-a)(p-b)(p-c)(p-d)}$,其中$p$是半周长
- 球缺：一个球被平面截下的一部分叫做球缺。截面叫做球缺的底面，垂直于截面的直径被截下的线段长叫做球缺的高。（设H是球缺高，R是球半径，d是底面半径） 球缺面积:$2\pi RH$ 球缺体积:$\frac{\pi}{3} * (3R-H) * H^2$ 球缺质心:在中轴线上和底面距离$c = \frac{(4R-H)H}{12R-4H} = \frac{(d^2+2H^2)H}{3d^2+4H^2}$
- 正多边形的边数为2^n且边数为不同的费马数的乘积形式的正多边形才可以在圆内用尺规画出(费马数：$2^{2^0}, 2^{2^1}, 2^{2^2} \dots 2^{2^k}$)

- 一类内切圆相关的问题

引例 如图 1，圆心为 A、B、C 的三个圆彼此外切，且均与直线 l 相切，设三个圆的半径分别为 a、b、$c(0<c<a<b)$. 证明：

$$\frac{1}{\sqrt{c}}=\frac{1}{\sqrt{a}}+\frac{1}{\sqrt{b}}. \qquad ①$$

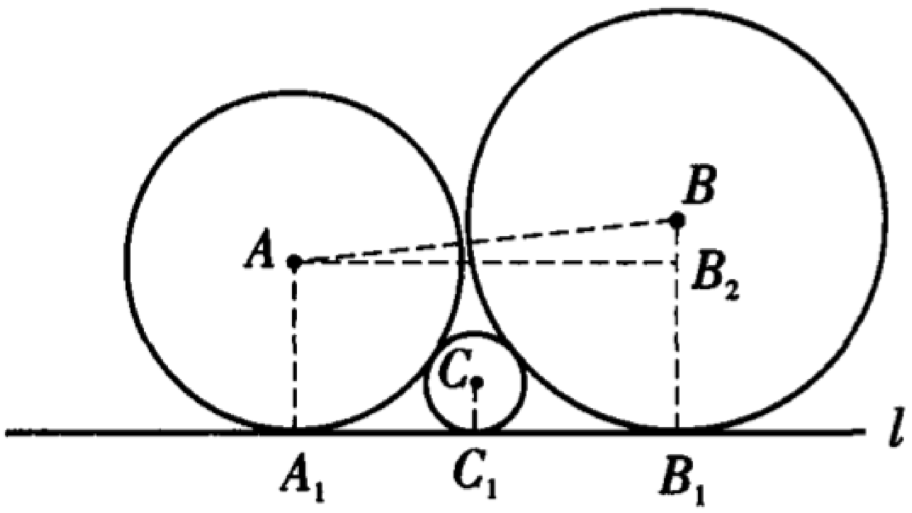


图 1

笛卡尔定理：若平面上半径为r_1,r_2,r_3,r_4的圆两两相切：

1. 若两两外切，则$(\sum_{i=1}^{4}\frac{1}{r_i})^2=2\sum_{i=1}^{4}\frac{1}{{r_i}^2}$
2. 若半径为r_1,r_2,r_3的圆内切于半径为r_4的大圆中，则$(\frac{1}{r_1}+\frac{1}{r_2}+\frac{1}{r_3}-\frac{1}{r_4})^2=2\sum_{i=1}^{4}\frac{1}{{r_i}^2}$
3. 推广到三维，若五个球的半径是$r_1,r_2\ldots r_5$，满足任意一个球与其他四个球外切，则$(\sum_{i=1}^{5}\frac{1}{r_i})^2=3\sum_{i=1}^{5}\frac{1}{{r_i}^2}$

常用模板

```cpp
#define maxn 100005
const double eps = 1e-7;
const double INF = 1e20;
const double pi = acos (-1.0);

int dcmp (double x) {
    if (fabs (x) < eps) return 0;
    return (x < 0 ? -1 : 1);
}
inline double sqr (double x) {return x*x;}
```

```
//*************点
struct Point {
    double x, y;
    Point (double _x = 0, double _y = 0):x(_x), y(_y) {}
    void input () {scanf ("%lf%lf", &x, &y);}
    void output () {printf ("%.2f %.2f\n", x, y);}
    bool operator == (const Point &b) const {
        return (dcmp (x-b.x) == 0 && dcmp (y-b.y) == 0);
    }
    bool operator < (const Point &b) const {
        return (dcmp (x-b.x) == 0 ? dcmp (y-b.y) < 0 : x < b.x);
    }
    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 * (double a) {
        return Point (x*a, y*a);
    }
    Point operator / (double a) {
        return Point (x/a, y/a);
    }
    double len2 () {//返回长度的平方
        return sqr (x) + sqr (y);
    }
    double len () {//返回长度
        return sqrt (len2 ());
    }
    Point change_len (double r) {//转化为长度为r的向量
        double l = len ();
        if (dcmp (l) == 0) return *this;//零向量返回自身
        r /= l;
        return Point (x*r, y*r);
    }
    Point rotate_left () {//顺时针旋转90度
        return Point (-y, x);
    }
    Point rotate_right () {//逆时针旋转90度
        return Point (y, -x);
    }
    Point rotate (Point p, double ang) {//绕点p逆时针旋转ang
        Point v = (*this)-p;
        double c = cos (ang), s = sin (ang);
        return Point (p.x + v.x*c - v.y*s, p.y + v.x*s + v.y*c);
    }
    Point normal () {//单位法向量
        double l = len ();
        return Point (-y/l, x/l);
    }
};

double cross (Point a, Point b) {//叉积
    return a.x*b.y-a.y*b.x;
}
double dot (Point a, Point b) {//点积
    return a.x*b.x + a.y*b.y;
}
double dis (Point a, Point b) {//两个点的距离
    Point p = b-a; return p.len ();
}
double degree_rad (double ang) {//角度转化为弧度
    return ang/180*pi;
}
double rad_degree (double rad) {//弧度转化为角度
    return rad/pi*180;
}
double rad (Point a, Point b) {//两个向量的夹角
    return fabs (atan2 (fabs (cross (a, b)), dot (a, b)) );
}
bool parallel (Point a, Point b) {//向量平行
    double p = rad (a, b);
    return dcmp (p) == 0 || dcmp (p-pi) == 0;
}

//************直线 线段
struct Line {
    Point s, e;//直线的两个点
    double k;//极角 范围[-pi,pi]
    Line () {}
    Line (Point _s, Point _e) {
        s = _s, e = _e;
        k = atan2 (e.y - s.y,e.x - s.x);
    }
    //ax+by+c = 0
    Line (double a, double b, double c) {
        if (dcmp (a) == 0) {
            s = Point (0, -c/b);
            e = Point (1, -c/b);
        }
        else if (dcmp (b) == 0) {
            s = Point (-c/a, 0);
            e = Point (-c/a, 1);
        }
        else {
            s = Point (0, -c/b);
            e = Point (1, (-c-a)/b);
        }
        get_angle ();
    }
    //一个点和倾斜角确定直线
    Line (Point p, double ang) {
        k = ang;
        s = p;
        if (dcmp (ang-pi/2) == 0) {
            e = s + Point (0, 1);
        }
        else
            e = s + Point (1, tan (ang));
    }
    void input () {
        s.input ();
        e.input ();
    }
    void output () {
        printf ("%.2f,%.2f %.2f,%.2f\n", s.x, s.y, e.x, e.y);
    }
    void adjust () {
        if (e < s) swap (e, s);
    }
    double length () {//求线段长度
        return dis (s, e);
    }
    void get_angle () {
        k = atan2 (e.y - s.y,e.x - s.x);
    }
    double angle () {//直线的倾斜角
        if (dcmp (k) < 0) k += pi;
        if (dcmp (k-pi) == 0) k -= pi;
        return k;
    }
    Point operator &(const Line &b)const {//直线的交点(保证存在)
        Point res = s;
        double t = (cross (s - b.s, b.s - b.e))/cross (s - e, b.s - b.e);
        res.x += (e.x - s.x)*t;
        res.y += (e.y - s.y)*t;
        return res;
    }
};

int relation (Point p, Line l) {//点和直线的关系
    //1:在左侧 2:在右侧 3:在直线上
    int c = dcmp (cross (p-l.s, l.e-l.s));
    if (c < 0) return 1;
    else if (c > 0) return 2;
    else return 3;
}

bool point_on_halfline (Point p, Line l) {//判断点在射线上
    int id = relation (p, l);
    if (id != 3) return 0;
    return dcmp (dot (p-l.s, l.e-l.s)) >= 0;
}

bool point_on_seg (Point p, Line l) {//判断点在线段上
    return dcmp (cross (p-l.s, l.e-l.s)) == 0 &&
    dcmp (dot (p-l.s, p-l.e)) <= 0;
    //如果忽略端点交点改成小于号就好了
}

bool parallel (Line a, Line b) {//直线平行
    return parallel (a.e-a.s, b.e-b.s);
}

int seg_cross_seg (Line a, Line v) {//线段相交判断
    //2:规范相交 1:不规范相交 0:不相交
    int d1 = dcmp (cross (a.e-a.s, v.s-a.s));
    int d2 = dcmp (cross (a.e-a.s, v.e-a.s));
    int d3 = dcmp (cross (v.e-v.s, a.s-v.s));
    int d4 = dcmp (cross (v.e-v.s, a.e-v.s));
    if ((d1^d2) == -2 && (d3^d4) == -2) return 2;
    return (d1 == 0 && dcmp (dot (v.s-a.s, v.s-a.e)) <= 0) ||
        (d2 == 0 && dcmp (dot (v.e-a.s, v.e-a.e)) <= 0) ||
        (d3 == 0 && dcmp (dot (a.s-v.s, a.s-v.e)) <= 0) ||
        (d4 == 0 && dcmp (dot (a.e-v.s, a.e-v.e)) <= 0);
}

int line_cross_seg (Line a, Line v) {//直线和线段相交判断 a直线v线段
    //2:规范相交 1:非规范相交 0:不相交
    int d1 = dcmp (cross (a.e-a.s, v.s-a.s));
    int d2 = dcmp (cross (a.e-a.s, v.e-a.s));
    if ((d1^d2) == -2) return 2;
    return (d1 == 0 || d2 == 0);
}

int line_cross_line (Line a, Line v) {//直线相交判断
    //0:平行 1:重合 2:相交
    if (parallel (a, v)) return relation (a.e, v) == 3;
    return 2;
}

Point line_intersection (Line a, Line v) {//直线交点
    //调用前确保有交点
    double a1 = cross (v.e-v.s, a.s-v.s);
    double a2 = cross (v.e-v.s, a.e-v.s);
    return Point ((a.s.x*a2-a.e.x*a1)/(a2-a1), (a.s.y*a2-a.e.y*a1)/(a2-a1));
}

int seg_intersectiong (Line a, Line b, Point &p) {//求线段交点
    //0:没有交点 1:规范相交 2:非规范相交
    //调用前确包只有一个交点
    int rel = seg_cross_seg (a, b);
    if (rel == 0) return 0;
    int cnt = 0;
    if (rel == 1) {
        if (point_on_seg (a.e, b)) p = a.e, cnt++;
        if (point_on_seg (a.s, b)) p = a.s, cnt++;
        if (point_on_seg (b.e, a)) p = b.e, cnt++;
        if (point_on_seg (b.s, a)) p = b.s, cnt++;
        return 2;
    }
    p = line_intersection (a, b);
    return 1;
}

double point_to_line (Point p, Line a) {//点到直线的距离
    return fabs (cross (p-a.s, a.e-a.s) / a.length ());
}

double point_to_seg (Point p, Line a) {//点到线段的距离
    if (dcmp (dot (p-a.s, a.e-a.s)) < 0 || dcmp (dot (p-a.e, a.s-a.e)) < 0)
        return min (dis (p, a.e), dis (p, a.s));
    return point_to_line (p, a);
}

Point projection (Point p, Line a) {//点在直线上的投影
    return a.s + (((a.e-a.s) * dot (a.e-a.s, p-a.s)) / (a.e-a.s).len2() );
}

Point symmetry (Point p, Line a) {//点关于直线的对称点
    Point q = projection (p, a);
    return Point (2*q.x-p.x, 2*q.y-p.y);
}

//***************圆
struct Circle {
    //圆心 半径
    Point p;
    double r;
    Circle () {}
    Circle (Point _p, double _r) : p(_p), r(_r) {}
    Circle (double a, double b, double _r) { //圆心坐标 半径确定圆
        p = Point (a, b);
        r = _r;
    }
    Circle (Point pt1, Point pt2, Point pt3) { //三点定圆 如果三点共线半径为-1
        double x1 = pt1.x, x2 = pt2.x, x3 = pt3.x;
        double y1 = pt1.y, y2 = pt2.y, y3 = pt3.y;
        double a = x1 - x2;
        double b = y1 - y2;
        double c = x1 - x3;
        double d = y1 - y3;
        double e = ((x1 * x1 - x2 * x2) + (y1 * y1 - y2 * y2)) / 2.0;
        double f = ((x1 * x1 - x3 * x3) + (y1 * y1 - y3 * y3)) / 2.0;
        double det = b * c - a * d;
        if (fabs (det) < eps) {
            r = -1;
            return Point(0,0);
        }
        double x0 = -(d * e - b * f) / det;
        double y0 = -(a * f - c * e) / det;
        r = hypot (x1 - x0, y1 - y0); P = Point (x0, y0);
    }
    void input () {
        p.input ();
        scanf ("%lf", &r);
    }
    void output () {
        p.output ();
        printf (" %.2f\n", r);
    }
    bool operator == (const Circle &a) const {
        return p == a.p && (dcmp (r-a.r) == 0);
    }
    double area () {//面积
        return pi*r*r;
    }
    double circumference () {//周长
        return 2*pi*r;
    }
    bool operator < (const Circle &a) const {
        return p < a.p || (p == a.p && r < a.r);
    }
};

int relation (Point p, Circle a) {//点和圆的关系
    //0:圆外 1:圆上 2:圆内
    double d = dis (p, a.p);
    if (dcmp (d-a.r) == 0) return 1;
    return (dcmp (d-a.r) < 0 ? 2 : 0);
}

int relation (Line a, Circle b) {//直线和圆的关系
    //0:相离 1:相切 2:相交
    double p = point_to_line (b.p, a);
    if (dcmp (p-b.r) == 0) return 1;
    return (dcmp (p-b.r) < 0 ? 2 : 0);
}

int relation (Circle a, Circle v) {//两圆的位置关系
    //1:内含 2:内切 3:相交 4:外切 5:相离
    double d = dis (a.p, v.p);
    if (dcmp (d-a.r-v.r) > 0) return 5;
    if (dcmp (d-a.r-v.r) == 0) return 4;
    double l = fabs (a.r-v.r);
    if (dcmp (d-a.r-v.r) < 0 && dcmp (d-l) > 0) return 3;
    if (dcmp (d-l) == 0) return 2;
    if (dcmp (d-l) < 0) return 1;
    return 0;
}

Circle out_circle (Point a, Point b, Point c) {//三角形外接圆
    Line u = Line ((a+b)/2, ((a+b)/2) + (b-a).rotate_left ());
    Line v = Line ((b+c)/2, ((b+c)/2) + (c-b).rotate_left ());
    Point p = line_intersection (u, v);
    double r = dis (p, a);
    return Circle (p, r);
}

Circle in_circle (Point a, Point b, Point c) {//三角形内切圆
    Line u, v;
    double m = atan2 (b.y-a.y, b.x-a.x), n = atan2 (c.y-a.y, c.x-a.x);
    u.s = a;
    u.e = u.s+Point (cos ((n+m)/2), sin ((n+m)/2));
    v.s = b;
    m = atan2 (a.y-b.y, a.x-b.x), n = atan2 (c.y-b.y, c.x-b.x);
    v.e = v.s + Point (cos ((n+m)/2), sin ((n+m)/2));
    Point p = line_intersection (u, v);
    double r = point_to_seg (p, Line (a, b));
    return Circle (p, r);
}

int circle_intersection (Circle a, Circle v, Point &p1, Point &p2) {//两个圆的交点
    //返回交点个数 交点保存在引用中
    int rel = relation (a, v);
    if (rel == 1 || rel == 5) return 0;
    double d = dis (a.p, v.p);
    double l = (d*d + a.r*a.r - v.r*v.r) / (2*d);
    double h = sqrt (a.r*a.
                 r - l*l);
    Point tmp = a.p + (v.p-a.p).change_len (l);
    p1 = tmp + ((v.p-a.p).rotate_left ().change_len (h));
    p2 = tmp + ((v.p-a.p).rotate_right ().change_len (h));
    if (rel == 2 || rel == 4) return 1;
    return 2;
}

int line_cirlce_intersection (Line v, Circle u, Point &p1, Point &p2) {//直线和圆的交点
    //返回交点个数 交点保存在引用中
    if (!relation (v, u)) return 0;
    Point a = projection (u.p, v);
    double d = point_to_line (u.p, v);
    d = sqrt (u.r*u.r - d*d);
    if (dcmp (d) == 0) {
        p1 = a, p2 = a;
        return 1;
    }
    p1 = a + (v.e-v.s).change_len (d);
    p2 = a - (v.e-v.s).change_len (d);
    return 2;
}

int get_circle (Point a, Point b, double r1, Circle &c1, Circle &c2) {//得到过ab半径为
r1的了两个圆
    //返回得到圆的个数 圆保存在两个引用中
    Circle x (a, r1), y (b, r1);
    int t = circle_intersection (x, y, c1.p, c2.p);
    if (!t) return 0;
    c1.r = c2.r = r1;
    return t;
}

int get_circle (Line u, Point q, double r1, Circle &c1, Circle &c2) {//得到和直线u相切
过点q 半径为r1的圆
    double d = point_to_line (q, u);
    if (dcmp (d-r1*2) > 0) return 0;
    if (dcmp (d) == 0) {
        c1.p = q + ((u.e-u.s).rotate_left ().change_len (r1));
        c2.p = q + ((u.e-u.s).rotate_right ().change_len (r1));
        c1.r = c2.r = r1;
        return 2;
    }
    Line u1 = Line (u.s + (u.e-u.s).rotate_left ().change_len (r1), u.e + (u.e-
u.s).rotate_left ().change_len (r1));
    Line u2 = Line (u.s + (u.e-u.s).rotate_right ().change_len (r1), u.e + (u.e-
u.s).rotate_right ().change_len (r1));
    Circle cc = Circle (q, r1);
    Point p1, p2;
    if (!line_cirlce_intersection (u1, cc, p1, p2))
        line_cirlce_intersection (u2, cc, p1, p2);
    c1 = Circle (p1, r1);
    if (p1 == p2) {
        c2 = c1;
        return 1;
    }
    c2 = Circle (p2, r1);
    return 2;
}

int get_circle (Line u, Line v, double r1, Circle &c1, Circle &c2, Circle &c3, Circle
&c4) {//和直线u,v相切 半径为r1的圆
    if (parallel (u, v)) return 0;
    Line u1 = Line (u.s + (u.e-u.s).rotate_left ().change_len (r1), u.e + (u.e-
u.s).rotate_left ().change_len (r1));
    Line u2 = Line (u.s + (u.e-u.s).rotate_right ().change_len (r1), u.e + (u.e-
u.s).rotate_right ().change_len (r1));
```

```
    Line v1 = Line (v.s + (v.e-v.s).rotate_left ().change_len (r1), v.e + (v.e-
v.s).rotate_left ().change_len (r1));
    Line v2 = Line (v.s + (v.e-v.s).rotate_right ().change_len (r1), v.e + (v.e-
v.s).rotate_right ().change_len (r1));
    c1.r = c2.r = c3.r = c4.r = r1;
    c1.p = line_intersection (u1, v1);
    c2.p = line_intersection (u1, v2);
    c3.p = line_intersection (u2, v1);
    c4.p = line_intersection (u2, v2);
    return 4;
}

int get_circle (Circle cx, Circle cy, double r1, Circle &c1, Circle &c2) {//和两个圆外
切 半径为r1的圆
    //确保两个圆外离
    Circle x (cx.p, r1+cx.r), y (cy.p, r1+cy.r);
    int t = circle_intersection (x, y, c1.p, c2.p);
    if (!t) return 0;
    c1.r = c2.r = r1;
    return t;
}

int tan_line (Point q, Circle a, Line &u, Line &v) {//过一点作圆切线
    int x = relation (q, a);
    if (x == 2) return 0;
    if (x == 1) {
        u = Line (q, q + (q-a.p).rotate_left ());
        v = u;
        return 1;
    }
    double d = dis (a.p, q);
    double l = a.r*a.r/d;
    double h = sqrt (a.r*a.r - l*l);
    u = Line (q, a.p + (q-a.p).change_len (l) + (q-a.p).rotate_left ().change_len
(h));
    v = Line (q, a.p + (q-a.p).change_len (l) + (q-a.p).rotate_right ().change_len
(h));
    return 2;
}

double area_circle (Circle a, Circle v) {//两圆相交面积
    int rel = relation (a, v);
    if (rel >= 4) return 0;
    if (rel <= 2) return min (a.area (), v.area ());
    double d = dis (a.p, v.p);
    double hf = (a.r+v.r+d)/2;
    double ss = 2*sqrt (hf*(hf-a.r)*(hf-v.r)*(hf-d));
    double a1 = acos ((a.r*a.r+d*d-v.r*v.r) / (2*a.r*d));
    a1 = a1*a.r*a.r;
    double a2 = acos ((v.r*v.r+d*d-a.r*a.r) / (2*v.r*d));
    a2 = a2*v.r*v.r;
    return a1+a2-ss;
}

double circle_traingle_area (Point a, Point b, Circle c) {//圆心三角形的面积
    //a.output (), b.output (), c.output ();
    Point p = c.p; double r = c.r; //cout << cross (p-a, p-b) << endl;
    if (dcmp (cross (p-a, p-b)) == 0) return 0;
    Point q[5];
    int len = 0;
    q[len++] = a;
    Line l(a, b);
    Point p1, p2;
    if (line_cirlce_intersection (l, c, q[1], q[2]) == 2) {
        if (dcmp (dot (a-q[1], b-q[1])) < 0) q[len++] = q[1];
        if (dcmp (dot (a-q[2], b-q[2])) < 0) q[len++] = q[2];
    }
    q[len++] = b;
    if (len == 4 && dcmp (dot (q[0]-q[1], q[2]-q[1])) > 0)
        swap (q[1], q[2]);
    double res = 0;
    for (int i = 0; i < len-1; i++) {
        if (relation (q[i], c) == 0 || relation (q[i+1], c) == 0) {
            double arg = rad (q[i]-p, q[i+1]-p);
            res += r*r*arg/2.0;
        }
        else {
            res += fabs (cross (q[i]-p, q[i+1]-p))/2;
        }
    } //cout << res << ".." << endl;
    return res;
}

//*************多边形

double polygon_area (Point *p, int n) {//多边形的有向面积,加上绝对值就是面积
    //n个点
    double area = 0;
    for (int i = 1; i < n-1; i++) {
        area += cross (p[i]-p[0], p[i+1]-p[0]);
    }
    return area/2;
}

int relation (Point q, Point *p, int n) {//点和多边形的关系(凸凹都可以)
    //0:外部 1:内部 2:边上 3:顶点
    for (int i = 0; i < n; i++) {
        if (p[i] == q)
            return 3;
    }
    for (int i = 0; i < n; i++) {
        if (point_on_seg (q, Line (p[i], p[(i+1)%n])))
```

```
            return 2;
    }
    int cnt = 0;
    for (int i = 0; i < n; i++) {
        int j = (i+1)%n;
        int k = dcmp (cross (q-p[j], p[i]-p[j]));
        int u = dcmp (p[i].y-q.y);
        int v = dcmp (p[j].y-q.y);
        if (k > 0 && u < 0 && v >= 0) cnt++;
        if (k < 0 && v < 0 && u >= 0) cnt--;
    }
    return cnt != 0;
}

int convex_cut (Line u, Point *p, int n, Point *po) {//直线切割多边形左侧
    //返回切割后多边形的数量
    int top = 0;
    for (int i = 0; i < n; i++) {
        int d1 = dcmp (cross (u.e-u.s, p[i]-u.s));
        int d2 = dcmp (cross (u.e-u.s, p[(i+1)%n]-u.s));
        if (d1 >= 0) po[top++] = p[i];
        if (d1*d2 < 0) po[top++] = line_intersection (u, Line (p[i], p[(i+1)%n]));
    }
    return top;
}

double convex_circumference (Point *p, int n) {//多边形的周长(凹凸都可以)
    double ans = 0;
    for (int i = 0; i < n; i++) {
        ans += dis (p[i], p[(i+1)%n]);
    }
    return ans;
}

double area_polygon_circle (Circle c, Point *p, int n) {//多边形和圆交面积
    double ans = 0;
    for (int i = 0; i < n; i++) {
        int j = (i+1)%n; //cout << i << " " << j << "//" << endl;
        if (dcmp (cross (p[j]-c.p, p[i]-c.p)) >= 0)
            ans += circle_traingle_area (p[i], p[j], c);
        else
            ans -= circle_traingle_area (p[i], p[j], c);
    }
    return fabs (ans);
}

Point centre_of_gravity (Point *p, int n) {//多边形的重心(凹凸都可以)
    double sum = 0.0, sumx = 0, sumy = 0;
    Point p1 = p[0], p2 = p[1], p3;
    for (int i = 2; i <= n-1; i++) {
        p3 = p[i];
        double area = cross (p2-p1, p3-p2)/2.0;
        sum += area;
        sumx += (p1.x+p2.x+p3.x)*area;
        sumy += (p1.y+p2.y+p3.y)*area;
        p2 = p3;
    }
    return Point (sumx/(3.0*sum), sumy/(3.0*sum));
}

int convex_hull (Point *p, Point *ch, int n) {//求凸包
        //所有的点集 凸包点集 点集的点数
    sort (p, p+n);
    int m = 0;
    for (int i = 0; i < n; i++) {
        while (m > 1 && cross (ch[m-1]-ch[m-2], p[i]-ch[m-1]) <= 0)
            m--;
        ch[m++] = p[i];
    }
    int k = m;
    for (int i = n-2; i >= 0; i--) {
        while (m > k && cross (ch[m-1]-ch[m-2], p[i]-ch[m-1]) <= 0)
            m--;
        ch[m++] = p[i];
    }
    if (n > 1)
        m--;
    return m;
}

double rotate_calipers (Point *p, int m) {//旋转卡壳 平面距离平方最远点对
    //如果求距离加sqrt即可
    double ans = 0;
    int cur = 1;
    for (int i = 0; i < m; i++) {
        while (cross (p[i]-p[(i+1)%m], p[(cur+1)%m]-p[cur]) < 0)
            cur = (cur+1)%m;
        ans = max (ans, max (dis (p[i], p[cur]), dis (p[(i+1)%m], p[(cur+1)%m])));
    }
    return ans;
}

//判断一个点是否在凸多边形内部或者边界上 复杂度 O(logn)
bool Compl_inside_convex(const Point & p,Point *con, int n)
{ //点  多边形数组（凸包顺序） 多边形数目
    if(n<3) return false;
    if(cross(p-con[0],con[1]-con[0])>eps) return false;   //不在边界的话变成>-eps
    if(cross(p-con[0],con[n-1]-con[0])<-eps) return false; //不在边界的话变成<eps
    int i=2,j=n-1;
    int line=-1;
    while(i<=j)
    {
        int mid=(i+j)>>1;
        if(cross(p-con[0],con[mid]-con[0])>-eps)
        {
            line=mid;
            j=mid-1;
        }
        else i=mid+1;
    }
    return cross(p-con[line-1],con[line]-con[line-1])<eps;  //不在边界的话变成<-eps
}


bool get_dir (Point *p, int n) {//得到多边形的时针顺序
    //0:顺时针 1:逆时针
    double sum = 0;
    for (int i = 0; i < n; i++) {
        sum += cross (p[i], p[(i+1)%n]);
    }
    if (dcmp (sum) > 0) return 1;
    return 0;
}

double poly_min_seg (Point *coin, int m) {
    //多边形的最短长度 可以是凹 复杂度O(n^2)
    //相当于硬币投影的最短长度
    double ans = INF;
    Point *ch;
    int cnt = convex_hull (coin, ch, m);
    for (int i = 0; i < cnt; i++) {
        int j = (i+1)%cnt;
        double tmp = 0;
        for (int k = 0; k < cnt; k++) if (k != i && k != j) {
            tmp = max (tmp, point_to_line (ch[k], Line (ch[i], ch[j])));
        }
        ans = min (ans, tmp);
    }
    return ans;
}

double poly_max_seg (Point *hole, int n) {
    //多边形的最长长度 可以是凹 复杂度O(n^3)
    //相当于孔洞提供的最长内部线段
    double ans = 0;
    vector <Point> gg;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) if (i != j) {
            Line cur (hole[i], hole[j]);
            gg.clear ();
            for (int x = 0; x < n; x++) {
                int y = (x+1)%n;
                int in1, in2;
                in1 = relation (hole[x], cur);
                in2 = relation (hole[y], cur);
                if (in1 == 3) gg.push_back (hole[x]);
                if (in2 == 3) gg.push_back (hole[y]);
                if (in1 == 3 || in2 == 3) {
                    continue;
                }
                int id = line_cross_seg (cur, Line (hole[x], hole[y]));
                if (id == 2) {
                    Point p = line_intersection (cur, Line (hole[x], hole[y]));
                }
            }
            sort (gg.begin (), gg.end ());
            int Max = gg.size ();
            double tmp = 0;
            for (int i = 0; i < Max-1; i++) {
                Point p = Point (gg[i].x+gg[i+1].x, gg[i].y+gg[i+1].y)/2;
                int id = relation (p, hole, n);
                if (id == 0) {
                    ans = max (ans, tmp);
                    tmp = 0;
                }
                else {
                    tmp += dis (gg[i], gg[i+1]);
                }
            }
            ans = max (ans, tmp);
        }
    }
    return ans;
}

//半平面交,直线的左边代表有效区域
bool HPIcmp (const Line &a, const Line &b) {
    if (fabs(a.k - b.k) > eps)
        return a.k < b.k;
    return cross (a.s - b.s, b.e - b.s) < 0;
}
Line Q[maxn];
void HPI(Line line[], int n, Point res[], int &resn) {
    //半平面数组 半平面个数 半平面交顶点 半平面交顶点个数
    for (int i = 0; i < n; i++) line[i].get_angle ();
    int tot = n;
    sort(line,line+n,HPIcmp);
    tot = 1;
    for(int i = 1;i < n;i++){
        if(fabs(line[i].k - line[i-1].k) > eps)
            line[tot++] = line[i];
    }
    int head = 0, tail = 1;
    Q[0] = line[0];
    Q[1] = line[1];
    resn = 0;
    for (int i = 2; i < tot; i++) {
        if (fabs(cross (Q[tail].e-Q[tail].s, Q[tail-1].e-Q[tail-1].s)) < eps ||
fabs(cross (Q[head].e-Q[head].s, Q[head+1].e-Q[head+1].s)) < eps)
            return;
        while(head < tail && (cross ((Q[tail]&Q[tail-1])-line[i].s, line[i].e-
line[i].s)) > eps) tail--;
        while(head < tail && (cross ((Q[head]&Q[head+1]) - line[i].s, line[i].e-
line[i].s)) > eps)
        head++;
        Q[++tail] = line[i];
    }
    while(head < tail && (cross ((Q[tail]&Q[tail-1]) - Q[head].s, Q[head].e-
Q[head].s)) > eps)
        tail--;
    while(head < tail && (cross ((Q[head]&Q[head-1]) -Q[tail].s, Q[tail].e-
Q[tail].e)) > eps)
        head++;
    if(tail <= head + 1)
        return;
    for(int i = head; i < tail; i++)
        res[resn++] = Q[i]&Q[i+1];
    if(head < tail - 1)
        res[resn++] = Q[head]&Q[tail];
}

//最大空凸包面积 最大的内部没有别的点(边界可以包含)的凸包面积 复杂度O(n^3)
Point List[maxn]; double opt[maxn][maxn]; int seq[maxn], len; double max_area;
bool cmp (Point a, Point b) {
    int tmp = dcmp (cross (a, b));
    if (tmp != 0) return tmp > 0;
    tmp = dcmp (b.len2 ()-a.len2 ());
    return tmp > 0;
}
bool lessY (Point a, Point b) {
    return dcmp (b.y-a.y) > 0 || (dcmp (b.y-a.y) == 0 && dcmp (b.x-a.x) > 0);
}
void solve (Point *p, int n, int vv) {
    int t, i, j, _len; double v;
    for (len = i = 0; i < n; i++) {
        if (lessY (p[vv], p[i])) List[len++] = p[i]-p[vv];
    }
    for (i = 0; i < len ;i++) {
        for (j = 0; j< len; j++) {
            opt[i][j] = 0;
        }
    }
    sort (List, List+len, cmp);
    for (t = 1; t < len; t++) {
        _len = 0;
        for (i = t-1; i >= 0 && dcmp (cross (List[t], List[i])) == 0; i--);
        while (i >= 0) {
            v = cross (List[i], List[t])/2;
            seq[_len++] = i;
            for (j = i-1; j >= 0 && dcmp (cross (List[i]-List[t], List[j]-List[t])) >
0; j--);
            if (j >= 0) v += opt[i][j];
            get_max (max_area, v);
            opt[t][i] = v;
            i = j;
        }
        for (i = _len-2; i >= 0; i--) {
            get_max (opt[t][seq[i]], opt[t][seq[i+1]]);
        }
    }
}
double max_empty_convexhull_area (Point *p, int n) {
    //点集 点数 求最大面积的时候调用这个接口
    max_area = 0;
    for (int i = 0; i < n; i++) solve (p, n, i);
    return max_area;
}
```

扫描线

三维几何

```
#define maxn 100005
const double eps = 1e-8;
const double INF = 1e20;
const double pi = acos (-1.0);

int dcmp (double x) {
    if (fabs (x) < eps) return 0;
    return (x < 0 ? -1 : 1);
}
struct Point {
    double x, y, z;
    Point (double _x = 0, double _y = 0, double _z = 0) : x(_x), y(_y), z(_z) {}
    void input () {
        scanf ("%lf%lf%lf", &x, &y, &z);
    }
    void output () {
        printf ("%.2f %.2f %.2f\n", x, y, z);
    }
    bool operator == (const Point &b) const {
        return dcmp (x-b.x) == 0 && dcmp (y-b.y) == 0 && dcmp (z-b.z) == 0;
    }
    double len2 () {
        return x*x+y*y+z*z;
    }
    double len () {
        return sqrt (x*x + y*y + z*z);
    }
    Point operator - (const Point &b) const {
        return Point (x-b.x, y-b.y, z-b.z);
    }
    Point operator + (const Point &b) const {
        return Point (x+b.x, y+b.y, z+b.z);
    }
    Point operator * (const double &k) const {
        return Point (x*k, y*k, z*k);
    }
    Point operator / (const double &k) const {
        return Point (x/k, y/k, z/k);
    }
    double operator * (const Point &b) const {
        return x*b.x+y*b.y+z*b.z;
    }
    Point operator ^ (const Point &b) const {
        return Point (y*b.z-z*b.y, z*b.x-x*b.z, x*b.y-y*b.x);
    }
    double rad (Point a, Point b) {//两向量的夹角
        Point p = (*this);
        return acos (((a-p)*(b-p)) / (a.distance (p)*b.distance (p)));
    }
    Point trunc (double r) {
        double l = len ();
        if (!dcmp (l)) return *this;
        r /= l;
        return Point (x*r, y*r, z*r);
    }
    double distance (Point p) {
        return Point (x-p.x, y-p.y, z-p.z).len ();
    }
};

struct Line {
    Point s, e;
    Line () {}
    Line (Point _s, Point _e) {
        s = _s, e = _e;
    }
    void input () {
        s.input ();
        e.input ();
    }
    void output () {
        cout << "line:" << endl;
        s.output ();
        e.output ();
    }
    double len () {
        return (e-s).len ();
    }
    double point_to_line (Point p) {//点到直线的距离
        return ((e-s)^(p-s)).len () / s.distance (e);
    }
    Point prog (Point p) {//点在直线上的投影
        return s+(((e-s)*((e-s)*(p-s))) / ((e-s).len2 ()));
    }
    bool Point_on_line (Point p) {//点在直线上
        return dcmp (((s-p)^(e-p)).len ()) == 0 && dcmp ((s-p)*(e-p)) == 0;
    }
};

struct Plane {
    Point a, b, c, o;
    Plane () {}
    Plane (Point _a, Point _b, Point _c) {
        a = _a, b = _b, c = _c;
        o = pvec ();
    }
    Point pvec () {//法向量
        return (b-a)^(c-a);
    }
    Plane go (Point p, double x) {//平面沿p方向前进x距离
        o = o.trunc (x);
        double ang = a.rad (a+o, p);
        if (ang >= pi/2) o = o*(-1);
        return Plane (a+o, b+o, c+o);
    }
    bool point_on_plane (Point p) {//点在平面上
        return dcmp ((p-a)*o) == 0;
    }
    double plane_angle (Plane f) {//两平面夹角
        return acos (o*f.o)/(o.len ()*f.o.len ());
    }
    int line_cross_plane (Line u, Point &p) {//直线和平面的交点
        double x = o*(u.e-a);
        double y = o*(u.s-a);
        double d = x-y;
        if (dcmp (d) == 0) return 0;
        p = ((u.s*x) - (u.e*y))/d;
        return 1;
    }
    double area () {//三角形面积
        double x1 = (a-b).len (), x2 = (a-c).len (), x3 = (b-c).len ();
        double p = (x1+x2+x3)/2;
        return sqrt (p*(p-x1)*(p-x2)*(p-x3));
    }
    Point point_to_plane (Point p) {
        Line u = Line (p, p+o);
        line_cross_plane (u, p);
        return p;
    }
    int plane_cross_plane (Plane f, Line &u) {//平面交线
        Point oo = o^f.o;
        Point v = o^oo;
        double d = fabs (f.o*v);
        if (dcmp (d) == 0)
            return 0;
        Point q = a-(v*(f.o*(f.a-a))/d);
        u = Line (q, q+oo);
        return 1;
    }
};
double ptoline(Point p,Point a,Point b){
    //点到直线的距离 直线用a,b表示
    Point res=cross(p-a,p-b);
    return res.len()/(b-a).len();
}
double ptoseg(Point p,Point a,Point b){
   //点到线段的距离 两端点为a,b
   if (a==b) return (p-a).len();
   Point v1=b-a,v2=p-a,v3=p-b;
   if (dcmp(dot(v1,v2))<0) return v2.len();
   else if (dcmp(dot(v1,v3))>0) return v3.len();
   else return cross(v1,v2).len()/v1.len();
}
Point Rotate(Point a,Point b,double angle){
   //向量/点a绕Ob向量逆时针旋转angle弧度
   //Ob向量的模长必须为1
   double x,y,z;
   x=(b.x*b.x+(1-b.x*b.x)*cos(angle))*a.x+(b.x*b.y*(1-cos(angle))-
b.z*sin(angle))*a.y+(b.x*b.z*(1-cos(angle))+b.y*sin(angle))*a.z;
   y=(b.y*b.y+(1-b.y*b.y)*cos(angle))*a.y+(b.z*b.y*(1-cos(angle))-
b.x*sin(angle))*a.z+(b.y*b.x*(1-cos(angle))+b.z*sin(angle))*a.x;
   z=(b.z*b.z+(1-b.z*b.z)*cos(angle))*a.z+(b.z*b.x*(1-cos(angle))-
b.y*sin(angle))*a.x+(b.y*b.z*(1-cos(angle))+b.x*sin(angle))*a.y;
   return Point(x,y,z);
}
```

平面最近点对

`HDU 1007`:求平面最近点对距离。

```
struct node {
    double x, y;
} p[maxn], tmp[maxn];
int cnt, n;
```

```
bool cmp1 (const node &a, const node &b) {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}

bool cmp2 (const node &a, const node &b) {
    return a.y < b.y;
}

double dis (node a, node b) {
    double xx = a.x-b.x, yy = a.y-b.y;
    return sqrt (xx*xx + yy*yy);
}

double solve (int l, int r) {
    double d = INF;
    if (l == r)
        return d;
    if (r-l == 1) {
        return dis (p[r], p[l]);
    }
    int mid = (l+r)>>1;
    d = min (solve (l, mid), solve (mid+1, r));
    int cnt = 0;
    for (int i = l; i <= r; i++) {
        if (fabs (p[mid].x-p[i].x) < d) {
            tmp[cnt++] = p[i];
        }
    }
    sort (tmp, tmp+cnt, cmp2);
    for (int i = 0; i < cnt; i++) {
        for (int j = i+1, tot = 1; j < cnt && tot <= 7 && tmp[j].y-tmp[i].y < d; j++)
{
            d = min (d, dis(tmp[i], tmp[j]));
        }
    }
    return d;
}

int main () {
    //freopen ("in", "r", stdin);
    while (scanf ("%d", &n) == 1 && n) {
        for (int i = 0; i < n; i++) {
            scanf ("%lf%lf", &p[i].x, &p[i].y);
        }
        sort (p, p+n, cmp1);
        printf ("%.2f\n", solve (0, n-1)/2.0);
    }
    return 0;
}
```

`HDU 4631`:给出n个点，从头开始插入点，把每次最近点对距离平方和加起来。

- 数据随机直接从所有的点开始删，记录最近点对。每次删掉了最近点对中的点后都重新做一次最近点对。

最近点对的数组下标[0,n-1]，调用Closest_Pair函数前先进行排序

```
struct Point
{
    int x,y;
    int id;
    int index;//在原始点中的下标位置
    Point(int _x = 0,int _y = 0,int _index = 0)
    {
        x = _x;
        y = _y;
        index = _index;
    }
}P[maxn];
long long dist(Point a,Point b)
{
    return ((long long)(a.x-b.x)*(a.x-b.x) + (long long)(a.y-b.y)*(a.y-b.y));
}
Point p[maxn];
Point tmpt[maxn], gg[maxn];
bool cmpxy(Point a,Point b)
{
    if(a.x != b.x)return a.x < b.x;
    else return a.y < b.y;
}
bool cmpy(Point a,Point b)
{
    return a.y < b.y;
}
pair<int,int> Closest_Pair(int left,int right)
{
    long long d = (1LL<<50);
    if(left == right)return make_pair(left,right);
    if(left + 1 == right)
        return make_pair(left,right);
    int mid = (left+right)/2;
    pair<int,int>pr1 = Closest_Pair(left,mid);
    pair<int,int>pr2 = Closest_Pair(mid+1,right);
    long long d1,d2;
    if(pr1.first == pr1.second)
        d1 = (1LL<<50);
    else d1 = dist(p[pr1.first],p[pr1.second]);

    if(pr2.first == pr2.second)
        d2 = (1LL<<50);
    else d2 = dist(p[pr2.first],p[pr2.second]);
    pair<int,int>ans;
    if(d1 < d2)ans = pr1;
    else ans = pr2;

    d = min(d1,d2);


    int k = 0;
    for(int i = left;i <= right;i++)
    {
        if((long long)(p[mid].x - p[i].x)*(p[mid].x - p[i].x) <= d)
            tmpt[k++] = p[i];
    }
    sort(tmpt,tmpt+k,cmpy);
    for(int i = 0;i <k;i++)
    {
        for(int j = i+1;j < k && (long long)(tmpt[j].y - tmpt[i].y)*(tmpt[j].y -
tmpt[i].y) < d;j++)
        {
            if(d > dist(tmpt[i],tmpt[j]))
            {
                d = dist(tmpt[i],tmpt[j]);
                ans = make_pair(tmpt[i].id,tmpt[j].id);
            }
        }
    }
    return ans;
}

int n;
long long ans, a, b, c, aa, bb, cc;
#define fi first
#define se second

int main () {
    //freopen ("more.in", "r", stdin);
    int t;
    scanf ("%d", &t);
    while (t--) {
        scanf ("%d%lld%lld%lld%lld%lld%lld", &n, &a, &b, &c, &aa, &bb, &cc);
        gg[0] = Point (0, 0, 0);
        for (int i = 1; i <= n; i++) {
            gg[i].x = (a*gg[i-1].x+b) % c;
            gg[i].y = (aa*gg[i-1].y+bb) % cc;
            gg[i].index = i;
            gg[i].id = 0;
        }
        int j = n;
        long long ans = 0;
        while (j > 1) {
            for (int i = 0; i < j; i++) p[i] = gg[i+1];
            sort (p, p+j, cmpxy);
            for (int i = 0; i < j; i++) p[i].id = i;
            pair <int, int> tmp = Closest_Pair (0, j-1);
            int Max = max (p[tmp.fi].index, p[tmp.se].index);
            ans += 1LL*(j-Max+1)*dist (p[tmp.fi], p[tmp.se]);
            j = Max-1;
        }
        printf ("%lld\n", ans);
    }
    return 0;
}
```

杂题

`JAG 2016 G`:给出n个点，用一条$x=i$将点分成$\leq i$和$>i$，求两边凸包面积和的最小值。

- 用水平序维护上凸壳和下凸壳的面积即可。可以用叉积避免精度误差。

```
#define maxn 100005

struct Point {
    long long x, y;
    Point (long long _x = 0, long long _y = 0) {
        x = _x, y = _y;
    }
    Point operator + (const Point &a) const {
        return Point (x+a.x, y+a.y);
    }
    Point operator - (const Point &a) const {
        return Point (x-a.x, y-a.y);
    }
    bool operator < (const Point &a) const {
        return x < a.x || (x == a.x && y < a.y);
    }
}p[maxn];
int n;

long long a[maxn], b[maxn], c[maxn], d[maxn];

long long cross (Point a, Point b) {
    return a.x*b.y - a.y*b.x;
}

int g1[maxn], m1;
int g2[maxn], m2;

long long solve () {
    m1 = m2 = 0;
    a[1] = b[1] = 0;
    g1[m1++] = 1;
    g2[m2++] = 1;
    int i = 1, j;
    while (i+1 <= n && p[i+1].x == p[1].x) {
        i++;
        a[i] = 0;
    }
    g1[m1++] = i;
    for (i++; i <= n; i = j+1) {
        j = i;
        while (j+1 <= n && p[j+1].x == p[i].x) j++;
        while (m1 > 1 && cross (p[j]-p[g1[m1-1]], p[g1[m1-1]]-p[g1[m1-2]]) <= 0) m1-
-;

        while (m2 > 1 && cross (p[g2[m2-1]]-p[g2[m2-2]], p[i]-p[g2[m2-1]]) <= 0) m2-
-;

        a[j] = a[g1[m1-1]] + cross (p[j]-p[1], p[g1[m1-1]]-p[1]);
        b[i] = b[g2[m2-1]] + cross (p[g2[m2-1]]-p[1], p[i]-p[1]);
        g1[m1++] = j, g2[m2++] = i;
    }


    m1 = m2 = 0;
    c[n] = d[n] = 0;
    g1[m1++] = n, g2[m2++] = n;
    i = n;
    while (i-1 >= 1 && p[i].x == p[i-1].x) {
        i--;
        d[i] = 0;
    }
    g2[m2++] = i;
    for (i--; i >= 1; i = j-1) {
        j = i;
        while (j >= 1 && p[j-1].x == p[i].x) j--;
        while (m1 > 1 && cross (p[g1[m1-1]]-p[g1[m1-2]], p[i]-p[g1[m1-1]]) <= 0) m1-
-;

        while (m2 > 1 && cross (p[j]-p[g2[m2-1]], p[g2[m2-1]]-p[g2[m2-2]]) <= 0) m2-
-;

        c[i] = c[g1[m1-1]] + cross (p[g1[m1-1]]-p[n], p[i]-p[n]);
        d[j] = d[g2[m2-1]] + cross (p[j]-p[n], p[g2[m2-1]]-p[n]);
        g1[m1++] = i;
        g2[m2++] = j;
    }

    long long ans = 8e18;
    int k;
    for (i = 1; i <= n; i = j+1) {
        j = i;
        while (j+1 <= n && p[j+1].x == p[i].x) j++;
        long long tmp1 = a[j]+b[i]+cross (p[i]-p[1], p[j]-p[1]);
        k = j+1;
        long long tmp2;
        if (k > n) {
            tmp2 = 0;
        }
        else {
            while (k+1 <= n && p[k+1].x == p[j+1].x) k++;
            tmp2 = c[k]+d[j+1]+cross (p[k]-p[n], p[j+1]-p[n]);
        }
        ans = min (ans, tmp1+tmp2);
    }
    return ans;
}

int main () {
    //freopen ("1.txt", "r", stdin);
    scanf ("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf ("%lld%lld", &p[i].x, &p[i].y);
    }
    sort (p+1, p+1+n);
    long long ans = solve ();
    cout << ans/2+ans%2 << endl;
    return 0;
}
```

`SPOJ RECTANGL`:平面和坐标轴平行矩形计数，保证无重点。

离散化后按照每一行的点数归类，high表示点数多余$\sqrt{n}$，反之为low。对于包含high的矩形暴力枚举行，扫其他所有点；对于仅由low构成的矩形，每一行暴力二重枚举。总复杂度$O(n\sqrt{n})$

```
pii p[maxn];
int n;
vector <int> X, Y; //用来离散化
vector <int> a[maxn], low, high; //每一行的点  点多的行  点少的行
int K; //分解点sqrt(n)
bool vis[maxn];

long long solve_high () {
    long long ans = 0;
    int highsize = high.size (), lowsize = low.size ();
    for (int i = 0; i < highsize; i++) {
        int id = high[i], sz = a[id].size (); Clear (vis, 0);
        for (int j = 0; j < sz; j++) vis[a[id][j]] = 1; //这个x坐标有点
        for (int ii = i+1; ii < highsize; ii++) {
            int idd = high[ii], szz = a[idd].size ();
            long long tmp = 0; //有多少个相同的x对
            for (int j = 0; j < szz; j++) if (vis[a[idd][j]]) tmp++;
            ans += tmp*(tmp-1)/2;
        }
        for (int ii = 0; ii < lowsize; ii++) {
            int idd = low[ii], szz = a[idd].size ();
            long long tmp = 0;
            for (int j = 0; j < szz; j++) if (vis[a[idd][j]]) tmp++;
            ans += tmp*(tmp-1)/2;
        }
    }
    return ans;
}

vector <pii> line; //所有low构成的线段
long long solve_low () {
    long long ans = 0;
    int lowsize = low.size ();
    line.clear ();
    for (int i = 0; i < lowsize; i++) {
        int id = low[i], sz = a[id].size ();
        for (int j = 0; j < sz; j++) {
            for (int k = j+1; k < sz; k++) {
                line.pb (make_pair (a[id][j], a[id][k]));
            }
        }
    }
    //统计一样的(a,b)对数  如果超时可以在这里优化
    sort (ALL (line)); long long tmp = 0; int sz = line.size ();
    for (int i = 0; i < sz; i++) {
        if (!i || line[i] != line[i-1]) {
            ans += tmp*(tmp-1)/2;
            tmp = 1;
        }
        else tmp++;
    }
    ans += tmp*(tmp-1)/2;
    return ans;
}

int main () {
    //Open ();
    scanf ("%d", &n); X.clear (); Y.clear ();
    for (int i = 1; i <= n; i++) {
        scanf ("%d%d", &p[i].fi, &p[i].se);
        X.pb (p[i].fi); Y.pb (p[i].se);
        a[i].clear ();
    }
    sort (ALL (X)); sort (ALL (Y));
    X.erase (unique (ALL (X)), X.end ()); Y.erase (unique (ALL (Y)), Y.end ());
    for (int i = 1; i <= n; i++) {
        p[i].fi = lower_bound (ALL (X), p[i].fi)-X.begin ()+1;
        p[i].se = lower_bound (ALL (Y), p[i].se)-Y.begin ()+1;
    }
    for (int i = 1; i <= n; i++) a[p[i].se].pb (p[i].fi);
    K = sqrt (n);
    for (int i = 1; i <= n; i++) if (a[i].size ()) {
        sort (ALL (a[i]));
        if (a[i].size () >= K) high.pb (i);
        else low.pb (i);
    }
    long long ans = 0;
    ans += solve_high (); //包含一个high行的矩形
    ans += solve_low (); //两个都是low行的矩形
    cout << ans << endl;
    return 0;
}
```

图论

一些结论

- 6元以及以上的图必然有三元独立集或者三元团。
- **平面图欧拉定理**：设G为任意的连通的平面图，则$v-e+f=2$，v是G的顶点数，e是G的边数，f是G的面数。
- 定义$f(edge)=number[edge.u]*number[edge.v], f(G)=\sum f(e)$，把总和k任意分配给所有点的最佳方案是找到最大团往里面平均分配。
- 定义n个点每个联通分量都是完全图的方案数是$g(x)$，有$g(x)=g(x-1)+g(x-2)-g(x-5)-g(x-7)+\ldots$，符号是++--++...。令$f(i)$其中第i项括号中减去的数，有$f(i)=3+2f(i-2)-f(i-4)$（前四项已经给出）。
- 奇环套上偶环会使得两个环上任意两点存在奇路径，偶环套上偶环会使得两个环上任意两点都是偶路径。

- 关于**树的直径**的一些性质：

1. 如果目前的直径是$<a,b>$，新增一个节点c之后，新的直径是$<a,c>,<a,b>,<b,c>$中的一个，也就是两次dfs求树直径的根据（双dfs求树直径的根据）；
2. 对于树上两种点，如果直径分别是$<a,b>$ 和 $<c,b>$，那么两种点合并以后新的直径是$max<a,c>,<a,d>,<b,c>,<b,d>.$

图上三元环计数

做法1：

- 统计每个点的度数;
- 度数$\leq\sqrt{m}$的分为第一类点，度数$>\sqrt{m}$的分为第二类
- 对于第一类，for每个点，然后for这个点的任意两条边，再判断这两条边的另一个端点是否连接 因为m条边最多每条边遍历一次，然后暴力的点的入度$\leq\sqrt{m}$，所以复杂度约为$O(m\sqrt{m})$
- 对于第二类，直接暴力任意三个点，判断这三个点是否构成环，因为这一类点的个数不会超过$\sqrt{m}$个，所以复杂度约为$O(\sqrt{m}^3)$. 判断两个点是否相连用hash。

`HDU 6184`:统计无向图四边形加一条斜边的结构数量。

转化成图上三元环问题，按照斜边的端点的类别分别统计。

```
int n, m, u, v;
vector <int> e[maxn];
bool mark[maxn];
```

```
int link[maxn];

void init() {
    memset(mark, 0, (n+5)*sizeof(mark[0]));
    memset(link, 0, (n+5)*sizeof(link[0]));
    for (int i = 1; i <= n; i++) {
        e[i].clear();
    }
}

void addEdge(int u, int v) {
    e[u].push_back(v);
}

int main() {
    memset (table.head, -1, sizeof table.head); table.tou.clear (); table.size = 0;
    while (~scanf("%d %d", &n, &m)) {
        table.init ();
        init();
        int b = sqrt(m);
        for (int i = 0; i < m; i++) {
            scanf("%d %d", &u, &v);
            addEdge(u, v);
            addEdge(v, u);
            table.add ((long long)u*n+v);
            table.add ((long long)v*n+u);
        }
        long long ans = 0;
        for (int i = 1; i <= n; i++) {
            int x = i;
            mark[i] = true;
            for (int j = 0; j < e[x].size(); j++) link[e[x][j]] = x;
            for (int j = 0; j < e[x].size(); j++) { //暴力枚举每一条边
                int y = e[x][j];
                long long sum = 0;
                if (mark[y]) {
                    continue;
                }
                if (e[y].size() <= b) {  //选择一个边少的进行枚举sqrt(m)
                    for (int k = 0; k < e[y].size(); k++) { //如果枚举的是y点的边，判断x
是否有连边
                        int z = e[y][k];
                        if (link[z] == x) sum++;
                    }
                } else {
                    for (int k = 0; k < e[x].size(); k++) { //如果枚举的是x点的边，判断是
否在set里
                        int z = e[x][k];
                        if (table.find((long long)z*n+y)) sum++;
                    }
                }
                ans += sum*(sum-1)/2;
            }

        }
        printf("%lld\n", ans);
    }
}
```

`BZOJ 3498` :求所有三元环值的和。三元环的值等于三个端点的最大值。

只需要认为定义图上边的方向即可。然后枚举的时候注意顺序和边的方向判断。

```
void addEdge(int u, int v) {
    e[u].push_back(v);
    table.add (1LL*u*n+v);
}

int main() {
    memset (table.head, -1, sizeof table.head); table.tou.clear (); table.size = 0;
    while (~scanf("%d %d", &n, &m)) {
        table.init ();
        init();
        int b = sqrt(m);
        for (int i = 1; i <= n; i++) scan (a[i]);
        for (int i = 0; i < m; i++) {
            scanf("%d %d", &u, &v);
            if (a[u] > a[v] || (a[u] == a[v] && u < v))
                addEdge(u, v);
            else
                addEdge(v, u);
        }
        long long ans = 0;
        for (int i = 1; i <= n; i++) {
            int x = i;
            for (int j = 0; j < e[x].size(); j++) link[e[x][j]] = x;
            for (int j = 0; j < e[x].size(); j++) { //暴力枚举每一条边
                int y = e[x][j];
                long long sum = 0;
                if (e[y].size() <= b) {  //选择一个边少的进行枚举sqrt(m)
                    for (int k = 0; k < e[y].size(); k++) { //如果枚举的是y点的边，判断x
是否有连边
                        int z = e[y][k];
                        if (link[z] == x) sum++;
                    }
                } else {
                    for (int k = 0; k < e[x].size(); k++) { //如果枚举的是x点的边，判断是
否在set里
                        int z = e[x][k];
                        if (table.find((long long)y*n+z)) sum++;
                    }
                }
                ans += sum*a[x];
            }

        }
        printf("%lld\n", ans);
    }
}
```

做法2

- 对所有边按照两端点的度数定向，度数小的往度数大的走，度数相同则按编号小到大走，这样定向后 可以保证是个有向无环图。因为要想定向存在环，则这个环上的点度数必须相同，由于保证了编号从小到大走 所以是不可能存在环的。
- 这样定向同时还保证了每个点的出度不超过$\sqrt{n}$，总复杂度$O(m\sqrt{n})$.

```
#include<bits/stdc++.h>
#define LL long long
#define P pair<int,int>
using namespace std;
const int N = 1e5 + 10;
set<LL> g;
int deg[N];
vector<pair<int,int> > G[N];
int vi[N];
int X[N * 2],Y[N * 2],cnt[N],pos[N];
int main(){
    int  n, m, u, v, Sz;
    while(scanf("%d%d",&n,&m) != EOF){
        Sz = sqrt(m);
        for(int i = 1;i <= n;i++){
            vi[i] = deg[i] = pos[i] = 0;
            G[i].clear();
        }
        g.clear();
        int tot = 0;
        for(int i = 0;i < m;i++){
            scanf("%d%d",&X[i],&Y[i]);
            u = X[i],v = Y[i];
            deg[u]++,deg[v]++;
        }
        for(int i = 0;i < m;i++){
            cnt[i] = 0;
            if(deg[X[i]] < deg[Y[i]]) G[X[i]].push_back(make_pair(Y[i],i));
            else if(deg[Y[i]] < deg[X[i]]) G[Y[i]].push_back(P(X[i],i));
            else{
                if(X[i] < Y[i]) G[X[i]].push_back(P(Y[i],i));
                else G[Y[i]].push_back(P(X[i],i));
            }
        }
        LL ans = 0;
        for(int i = 0;i < m;i++){
            u = X[i],v = Y[i];
            for(auto vp:G[u]) pos[vp.first] = vp.second,vi[vp.first] = i + 1;
            for(auto vp:G[v]){
                int vv = vp.first;
                if(vi[vv] == i + 1){
                    cnt[i]++;
                    cnt[pos[vv]]++;
                    cnt[vp.second]++;
                }
            }
        }
        for(int i = 0;i < m;i++) ans += 1LL * cnt[i] * (cnt[i] - 1) / 2;
        printf("%lld\n",ans);
    }
    return 0;
}
```

最大团

- Bron–Kerbosch搜索最大团。

```
int n;
namespace MaxClique {
    bool g[maxn][maxn];
    int ans, adj[maxn][maxn], ma[maxn];
    bool dfs(int now,int dep) {
        int i, j, u, v, poi;
        for (int i = 0; i < now; i++) {
            if (dep+now-i <= ans) return false;
            u = adj[dep][i], poi = 0;
            if (dep+ma[u] <= ans) return false;
            for (int j = i+1; j < now; j++) if (v=adj[dep][j], g[u][v]) adj[dep+1]
[poi++] = v;
            if (dfs(poi,dep+1)) return true;
        }
        if (dep > ans) {
            ans = dep; // upd max clique
            return true;
        }
        return false;
    }
    int work() { //返回最大团的点数
        int i, j, poi;
        ans = 0;
        for (int i = n-1; i >= 0; i--) {
            poi = 0;
            for (int j = i+1; j < n; j++) if (g[i][j]) adj[1][poi++] = j;
            dfs(poi,1);
```

```
                ma[i] = ans;
            }
            return ans;
        }
    }
```

生成树

最小生成树

定理一：如果 A, B 同为 G 的最小生成树，且 A 的边权从小到大为 $w(a_1), w(a_2), w(a_3), \cdots w(a_n)$，$B$ 的边权从小到大为 $w(b_1), w(b_2), w(b_3), \cdots w(b_n)$，则有 $w(a_i) = w(b_i)$。 **定理二**：如果 A, B 同为 G 的最小生成树，如果 A, B 都从零开始从小到大加边（A 加 A 的边，B 加 B 的边）的话，每种权值加完后图的联通性相同。 **定理三**：如果在最小生成树 A 中权值为 v 的边有 k 条，用任意 k 条权值为 v 的边替换 A 中的权为 v 的边且**不产生环**的方案都是一棵合法最小生成树。

- 求最小生成树：用并查集维护。

```
struct node {
    int u, v, w;
    bool operator < (const node &a) const {
        return w < a.w;
    }
}edge[maxm];

int fa[maxn];
int Find (int x) {return x == fa[x] ? fa[x] : fa[x] = Find (fa[x]);}
long long MST () {
    sort (edge, edge+m);
    for (int i = 1; i <= n; i++) fa[i] = i;
    long long ans = 0;
    for (int i = 0; i < m; i++) {
        int u = edge[i].u, v = edge[i].v;
        int p1 = Find (u), p2 = Find (v);
        if (p1 != p2) {
            fa[p1] = p2;
            ans += edge[i].w;
        }
    }
    return ans;
}
```

- 最小生成树计数（BZOJ 1016）：按照边的权值分类，每类边的数量必然数固定的。

最小基环森林

`codeforces 875F`:一种特殊二分图的最大权匹配，右每个点都仅和左边两个点连边且权值相同，求最大权匹配。

如果右边一个点和左边的u, v均有连边，考虑连$< u, v, w >$的无向边，就是要将这些边定向使得所有点度数最多为1。最后就是要定下一个最大基环森林（一个分支最多只能有一个基环）。 并查集维护每个分支是否有基环即可。

```
struct node {
    int u, v, w;
    bool operator < (const node &a) const {
        return w < a.w;
    }
};
vector <node> edge;
int n, m, fa[maxn], is[maxn] = {0};
int Find (int x) {return x == fa[x] ? fa[x] : fa[x] = Find (fa[x]);}
long long solve () {
    sort (ALL (edge)); reverse (ALL (edge));
    long long ans = 0;
    int sz = edge.size ();
    for (int i = 0; i < sz; i++) {
        int u = edge[i].u, v = edge[i].v, w = edge[i].w;
        int p1 = Find (u), p2 = Find (v);
        if (p1 == p2) {
            if (!is[p1]) {
                ans += edge[i].w;
                is[p1] = 1;
            }
        }
        else if (p1 != p2) {
            if (is[p1] && is[p2]) continue;
            fa[p1] = p2;
            ans += w;
            if (is[p2] || is[p1]) is[p2] = 1;
        }
    }
    return ans;
}

int main () {
    cin >> n >> m; edge.clear ();
    for (int i = 1; i <= n; i++) fa[i] = i;
    for (int i = 1; i <= m; i++) {
        int u, v, w; cin >> u >> v >> w;
        edge.pb ((node) {u, v, w});
    }
    long long ans = solve ();
    cout << ans << endl;
    return 0;
}
```

生成树计数(matrix-tree定理)

- Kirchhoff矩阵:矩阵对角线是节点的度数，其他元素如果存在这条边就是-1，否则就是0。矩阵的**n-1**阶主子式的行列式的值就是生成树的个数。复杂度$O(n^3)$.

```
long long det () {    //按列化为下三角
```

```
    //答案对mod取模
    long long res = 1;
    for (int i = 0; i < n; ++i) {
        if (!a[i][i]) {
            bool flag = false;
            for (int j = i + 1; j < n; ++j) {
                if (a[j][i]) {
                    flag = true;
                    for (int k = i; k < n; ++k) {
                        swap (a[i][k], a[j][k]);
                    }
                    res = -res;
                    break;
                }
            }
            if (!flag) {
                return 0;
            }
        }
        for (int j = i + 1; j < n; ++j) {
            while (a[j][i]) {
                long long t = a[i][i] / a[j][i];
                for (int k = i; k < n; ++k) {
                    a[i][k] = (a[i][k] - t * a[j][k]) % mod;
                    swap (a[i][k], a[j][k]);
                }
                res = -res;
            }
        }
        res *= a[i][i];
        res %= mod;
    }
    return (res+mod)%mod;
}
```

最优比率生成树

- 边权有两个值a和b，选取最生成树使得sum(a)/sum(b)最大。

1. 二分：$\frac{\sum a_i}{\sum b_i} \leq x$，说明要选取一些边使得$\sum(a_i - b_i) \leq 0$，按照$a_i - x * b_i$为边权作最小生成树，二分即可；
2. 迭代：随便选一个结果，按照上面的方法求出最小生成树，然后用最小生成树对应的结果迭代更新。

`POJ 2728`：有n个村庄，村庄在不同坐标和海拔，现在要对所有村庄供水，建造水管距离为坐标之间的欧几里德距离，费用为海拔之差，现在要求方案使得费用与距离的比值最小。

- 二分

```
#define maxn 1111
#define eps 1e-5
#define INF 1e6
```

```
#define maxn 1111

int n;
double mp[maxn][2], h[maxn];
double d[maxn];
bool vis[maxn];

double dis (int i, int j) {
    double xx = mp[i][0]-mp[j][0], yy = mp[i][1]-mp[j][1];
    return sqrt (xx*xx + yy*yy);
}

bool ok (double cur) {
    memset (vis, 0, sizeof vis);
    for (int i = 1; i < n; i++) {
        d[i] = abs (h[i]-h[0])-cur*dis (i, 0);
    }
    vis[0] = 1;
    double ans = 0;
    for (int i = 1; i < n; i++) {
        double y = INF;
        int id;
        for (int j = 0; j < n; j++) if (!vis[j] && d[j] < y) {
            y = d[j];
            id = j;
        }
        vis[id] = 1;
        ans += y;
        for (int j = 0; j < n; j++) if (!vis[j]) {
            d[j] = min (d[j], abs (h[id]-h[j])-cur*dis (j, id));
        }
    }
    return ans <= 0;
}

int main () {
    //freopen ("in.txt", "r", stdin);
    while (scanf ("%d", &n) == 1 && n) {
        for (int i = 0; i < n; i++) {
            scanf ("%lf%lf%lf", &mp[i][0], &mp[i][1], &h[i]);
        }
        double ans, l = 0, r = INF;
        while (r-l > eps) {
            double mid = (l+r)/2;
            if (ok (mid))
                r = mid;
            else
                l = mid;
        }
        if (ok (l)) {
            ans = l;
        }
        else ans = r;
        printf ("%.3f\n", ans);
    }
    return 0;
}
```

- 迭代

```
#define eps 1e-5
#define INF 1e6

int n;
double mp[maxn][2], h[maxn];
double d[maxn];
bool vis[maxn];
int pre[maxn];

double dis (int i, int j) {
    double xx = mp[i][0]-mp[j][0], yy = mp[i][1]-mp[j][1];
    return sqrt (xx*xx + yy*yy);
}

double prim (double cur) {
    double a = 0, b = 0;
    memset (vis, 0, sizeof vis);
    for (int i = 1; i < n; i++) {
        d[i] = abs (h[i]-h[0])-cur*dis (i, 0);
        pre[i] = 0;
    }
    vis[0] = 1; pre[0] = 0;
    double ans = 0;
    for (int i = 1; i < n; i++) {
        double y = INF;
        int id;
        for (int j = 0; j < n; j++) if (!vis[j] && d[j] < y) {
            y = d[j];
            id = j;
        }
        vis[id] = 1;
        ans += y;
        a += abs (h[id]-h[pre[id]]);
        b += dis (id, pre[id]);
        for (int j = 0; j < n; j++) if (!vis[j] && d[j] > abs (h[id]-h[j])-cur*dis
(j, id)) {
            d[j] = abs (h[id]-h[j])-cur*dis (j, id);
            pre[j] = id;
        }
    }
    return a/b;
}

int main () {
    while (scanf ("%d", &n) == 1 && n) {
        for (int i = 0; i < n; i++) {
            scanf ("%lf%lf%lf", &mp[i][0], &mp[i][1], &h[i]);
        }
        double a = 0, b;
        while(1)
        {
            b = prim(a);
            if(fabs(a - b) < eps) break;
            a = b;
        }
        printf("%.3f\n", b);
    }
    return 0;
}
```

最小树形图

朱刘算法: 从单个源点出发的最小树形图，复杂度$O(nm)$。

```
struct Edge {
    int u, v, cost;
}edge[maxm];
int pre[maxn], id[maxn], vis[maxn], in[maxn];
int zhuliu (int root, int n, int m) {
    int res = 0,u,v;
    while (1) {
        for (int i = 0; i < n; i++)
            in[i] = INF;
        for (int i = 0; i < m; i++)
            if (edge[i].u != edge[i].v && edge[i].cost < in[edge[i].v]) {
                pre[edge[i].v] = edge[i].u;
                in[edge[i].v] = edge[i].cost;
            }
        for (int i = 0;i < n;i++)
        if (i != root && in[i] == INF)
        return -1;//不存在最小树形图
        int tn = 0;
        memset (id, -1, sizeof(id));
        memset (vis, -1, sizeof(vis));
        in[root] = 0;
        for (int i = 0; i < n; i++) {
            res += in[i];
            v = i;
            while ( vis[v] != i && id[v] == -1 && v != root) {
                vis[v] = i;
                v = pre[v];
            }
            if ( v != root && id[v] == -1 ) {
                for (int u = pre[v]; u != v; u = pre[u])
                    id[u] = tn;
                id[v] = tn++;
            }
        }
        if (tn == 0) break;//没有有向环
        for (int i = 0; i < n;i++)
        if (id[i] == -1)
        id[i] = tn++;
        for (int i = 0; i < m;) {
            v = edge[i].v;
            edge[i].u = id[edge[i].u];
            edge[i].v = id[edge[i].v];
            if (edge[i].u != edge[i].v)
                edge[i++].cost -= in[v];
            else
                swap(edge[i],edge[--m]);
        }
        n = tn;
        root = id[root];
    }
    return res;
}
```

最短路&&差分约束

dijstra

- 复杂度$O(n^2)$

```
void dij (int from, long long *d1) {
    memset (vis, 0, sizeof vis);
    for (int i = 1; i <= n; i++)
        d1[i] = (i == from ? 0 : INF);
    for (int i = 1; i <= n; i++) {
        long long cur = INF;
        int x;
        for (int j = 1; j <= n; j++) if (!vis[j] && d1[j] <= cur) {
            cur = d1[j];
            x = j;
        }
        vis[x] = 1;
        for (int j = 1; j <= n; j++) {
            d1[j] = min (d1[j], d1[x]+mp[x][j]);
        }
    }
    return ;
}
```

- 堆优化,复杂度$O(nlgm)$

```
bool vis[maxn];
long long d[maxn];
struct E {
    int v; long long w;
    bool operator < (const E &_) const {
        return w > _.w;
    }
};
void dij (int s, int n) {//源点 总点数
    for (int i = 1; i <= n; i++) {
        d[i] = INF;
        vis[i] = 0;
    }
    vis[s] = 1; d[s] = 0;
    priority_queue <E> q;
    for (int i = head[s]; i+1; i = edge[i].next) {
        q.push ((E) {edge[i].v, edge[i].w});
    }
    while (!q.empty ()) {
        E tmp = q.top (); q.pop ();
        int v = tmp.v; long long w = tmp.w; //cout << v << endl;
        if (vis[tmp.v]) continue;
        d[v] = w;
        vis[v] = 1;
        for (int i = head[v]; i+1; i = edge[i].next) {
            if (!vis[edge[i].v])
                q.push ((E) {edge[i].v, d[v]+edge[i].w});
        }
    }
}
```

SPFA

复杂度O(k*e)，k为每个点的入队次数

```
//返回1表示没有负环 否则表示存在负环
//如果要判断全图是否有负环，初始化时将全部点入队列,否则初始化源点入队。
bool vis[maxn];
int top, num[maxn];
double d[maxn];
bool spfa (int n) {
    queue <int> q;
    while (!q.empty ()) q.pop ();
    memset (num, 0, sizeof num);
    for (int i = 0; i < n; i++) {
        vis[i] = 1;
        d[i] = 0;
        q.push (i);
        num[i] = 1;
    }

    while (!q.empty ()) {
        int u = q.front (); q.pop ();
        vis[u] = 0;
        for (int i = head[u]; i != -1; i = edge[i].next) {
            int v = edge[i].v;
            if (d[v] > d[u]+edge[i].w) {
                d[v] = d[u]+edge[i].w;
                if (!vis[v]) {
                    vis[v] = 1;
                    q.push (v);
                    if (++num[v] > n)
                        return 0;
                }
            }
        }
    }
    return 1;
}
```

floyd

```
//初始化：mp[i][i]=初始值(0或者INF，根据题意来)
//如果<i,j>没有边 dp[i][j]=INF(小心溢出)
void floyd () {
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                mp[i][j] = min (mp[i][j], mp[i][k]+mp[k][j]);
            }
        }
    }
}
```

最小字典序单源最短路

`HDU 4871`:前半题是求最短路径下的一个从源点出发的字典序最小的生成树。

```
int dis[maxn], vis[maxn]; queue <int> q; vector <int> add[maxn];
void build_tree (int u, int fa) {
    int sz = G[u].size (); vis[u] = 1;
    for (int i = 0; i < sz; i++) {
        int v = G[u][i].fi, w = G[u][i].se; if (v == fa || vis[v]) continue;
        if (dis[u]+w == dis[v]) {
            add_edge (u, v, w); add_edge (v, u, w);
            add[u].pb (v);
            vis[v] = 1;
        }
    }
    sz = add[u].size ();
    for (int i = 0; i < sz; i++) {
        build_tree (add[u][i], u);
    }
}
void spfa (int s) {
    for (int i = 1; i <= n; i++) sort (ALL (G[i]));
    while (!q.empty ()) q.pop ();
    Clear (vis, 0);
    for (int i = 1; i <= n; i++) dis[i] = INF;
    q.push (s); vis[s] = 1; dis[s] = 0;
    while (!q.empty ()) {
        int u = q.front (); q.pop ();
        vis[u] = 0;
        int sz = G[u].size ();
        for (int i = 0; i < sz; i++) {
            int v = G[u][i].fi, w = G[u][i].se;
            if (dis[u]+w < dis[v]) {
                dis[v] = dis[u] + w;
                if (!vis[v]) {
                    vis[v] = 1;
                    q.push (v);
                }
            }
        }
    }
    Clear (vis, 0);
    build_tree (1, 0);
}
```

最优比率环

`POJ 3621`:找出一个环, 最大化$\frac{\sum v_i}{\sum E_i}$.

假设答案是ans.那么有$\frac{\sum v_i}{\sum E_i} = ans$也就是$\sum v_i - ans\sum E_i = 0$.因为是一个环,所以环上每个点都被算了两次,所以可以给每条边对应一个F_i表示边上两个节点权值和的一半,所以所求就是$\sum F_i - ans\sum E_i = 0$.二分枚举ans, 如果找到一个负环,说明ans偏小,否则ans偏大.

```
int n, m;
int val[maxn];
struct node {
    int u, v, num, next;
    double w;
}edge[maxm];
int head[maxn], cnt;
```

```
void add_edge (int u, int v, int w) {
    edge[cnt].num = w;
    edge[cnt].u = u, edge[cnt].v = v, edge[cnt].next = head[u], head[u] = cnt++;
}

bool vis[maxn];
int top, num[maxn];
double d[maxn];

bool spfa (double ans) {
    queue <int> q; while (!q.empty ()) q.pop ();
    for (int i = 1; i <= n; i++) {
        vis[i] = 1;
        d[i] = 0;
        q.push (i);
        num[i] = 1;
    }
    for (int i = 0; i < cnt; i++) {
        node &e = edge[i];
        e.w = (val[e.u]+val[e.v])/2.0-ans*e.num;
    }
    while (!q.empty ()) {
        int u = q.front (); q.pop ();
        vis[u] = 0;
        for (int i = head[u]; i+1; i = edge[i].next) {
            int v = edge[i].v;
            if (d[v] < d[u]+edge[i].w) {
                d[v] = d[u]+edge[i].w;
                if (!vis[v]) {
                    vis[v] = 1;
                    q.push (v);
                    if (++num[v] > n)
                        return 1;
                }
            }
        }
    }
    return 0;
}

int main () {
    while (scanf ("%d%d", &n, &m) == 2) {
        double l = 0, r = 0;
        for (int i = 1; i <= n; i++) {
            scanf ("%d", &val[i]);
            r += val[i];
        }
        Clear (head, -1);
        cnt = 0;
        for (int i = 1; i <= m; i++) {
            int u, v, w; scanf ("%d%d%d", &u, &v, &w);
            add_edge (u, v, w);
        }
        while (r-l > 1e-5) {
            double mid = (l+r)/2;
            if (spfa (mid)) l = mid;
            else r = mid;
        }
        printf ("%.2f\n", l);
    }
    return 0;
}
```

差分约束

一系列$x_i - x_j \leq a$的条件的组合，方法就是建边$x_j - > x_i$，边权是a。

`HDU 1384`: 求一个最小size的集合, 使得满足n个条件, 每个条件描述在一个闭区间中至少有多少个元素在集合中.

用t_i表示[0,i-1]区间有多少个元素在集合中, 然后相当于要满足的条件有: $f(x) = \begin{cases} t_a - t_{b+1} \leq -c \\ 0 \leq t_{i+1} - t_i \leq 1 \end{cases}$ 把上面的式子转化一下成差分式子, 然后从50001这个点跑一次最短路, 取1这个点的最短路的绝对值就好. 这个含义就是对于每一个点, 都满足差分式子, 因为如果不满足肯定还能更新最短路.

```
struct node {
    int v, next;
    int w;
}edge[maxm];
int cnt, head[maxn];
int n;

void add_edge (int u, int v, int w) {
    edge[cnt].v = v, edge[cnt].next = head[u], edge[cnt].w = w, head[u] = cnt++;
}
int d[maxn];

bool vis[maxn];
int top, num[maxn];
bool spfa (int start, int n) {
    memset (vis, 0, sizeof vis);
    for (int i = 1; i <= n; i++) {
        d[i] = INF;
    }
    vis[start] = 1;
    d[start] = 0;
    queue <int> q;
    while (!q.empty ()) q.pop ();
    q.push (start);
    memset (num, 0, sizeof num);
    num[start] = 1;
    while (!q.empty ()) {
        int u = q.front (); q.pop ();
        vis[u] = 0;
        for (int i = head[u]; i != -1; i = edge[i].next) {
            int v = edge[i].v;
            if (d[v]>d[u]+edge[i].w) {
                d[v] = d[u]+edge[i].w;
                if (!vis[v]) {
                    vis[v] = 1;
                    q.push (v);
                    if (++num[v] > n)
                        return 0;
                }
            }
        }
    }
    return 1;
}

int main () {
    while (scanf ("%d", &n) == 1) {
        memset (head, -1, sizeof head);
        cnt = 0;
        for (int i = 1; i <= n; i++) {
            int a, b, c;
            scanf ("%d%d%d", &a, &b, &c);
            add_edge (b+1, a, -c);
        }
        for (int i = 1; i <= 50000; i++) {
            add_edge (i+1, i, 0);
            add_edge (i, i+1, 1);
        }
        int ans = spfa (50001, 50001);
        printf ("%d\n", abs (d[1]));
    }
    return 0;
}
```

`HDU 3666`: 给出一个矩阵, 问是否存在两个序列$a_1, a_2 \dots a_n$和$b_1, b_2 \dots b_m$使得$a_i \times c_{i,j}/b_j \in [L, U]$.

整理一下条件就是要判断是否存在两个数列使得 $$ f(x)=\left{ \begin{aligned} a_i\times c*{i,j}\leq U\times b_j \ L\times b_j \leq a_i\times c*{i, j} \end{aligned} \right.

$$\text{然后用}log\text{去乘号：}$$

f(x)=\left\{

$$\begin{aligned} log(a_i) - log(b_j) \leq log(U/c_{i,j}) \\ log(b_j) - log(a_i) \leq log(c_{i,j}/L) \end{aligned}$$

\right.
$

然后判断负环就好了. 如果判定条件是入队次数大于n会超时, 用另一种判断:$\text{入队次数} \geq \sqrt{n+m}$.

```cpp
int n, m;
double l, u;
double c[411][411];
struct node {
    int v, next;
    double w;
}edge[maxm];
int head[maxn], cnt;

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

void init () {
    memset (head, -1, sizeof head);
    cnt = 0;
}

void add_edge (int u, int v, double w) {
    edge[cnt].v = v, edge[cnt].next = head[u], edge[cnt].w = w, head[u] = cnt++;
}

bool vis[maxn];
int top, num[maxn];
double d[maxn];
bool spfa (int start, int n, int m) {
    memset (vis, 0, sizeof vis);
    for (int i = 1; i <= n; i++) {
        d[i] = INF;
    }
    vis[start] = 1;
    d[start] = 0;
    queue <int> q;
    while (!q.empty ()) q.pop ();
    q.push (start);
    memset (num, 0, sizeof num);
    num[start] = 1;
    while (!q.empty ()) {
        int u = q.front (); q.pop ();
        vis[u] = 0;
        for (int i = head[u]; i != -1; i = edge[i].next) {
            int v = edge[i].v;
            if (d[v]>d[u]+edge[i].w) {
                d[v] = d[u]+edge[i].w;
                if (!vis[v]) {
                    vis[v] = 1;
                    q.push (v);
                    if (++num[v] > (int)sqrt ((1.0*n+m))/*n*/)
                        return 0;
                }
            }
        }
    }
    return 1;
}

int main () {
    while (scanf ("%d%d", &n, &m) == 2) {
        scanf ("%lf%lf", &l, &u);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                scanf ("%lf", &c[i][j]);
            }
        }
        init ();
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                add_edge (i, n+j, log10 (u/c[i][j]));
                add_edge (n+j, i, log10 (c[i][j]/l));
            }
        }
        if (spfa (0, n+m, 2*(n+m)))
            printf ("YES\n");
        else
            printf ("NO\n");
    }
    return 0;
}
```

同余最短路

`HDU 6071`: 1,2,3,4构成一个环，告诉每条边的长度，求一个从2出发回到2的路径，使得总长度不小于K且最小。

假设2到1的距离为w，现在有一种从2回到2总长度为s的方案，那么通过来回走必然有$s+k*w$的方案。用$d[i][j]$表示从2出发到i，走的长度模w为j的最短路程。然后就可以直接spfa，最后在$dp[2][j]$中遍历，将长度补上。

```
long long a[4], K;
struct node {
    int u, v, next;
    long long w;
}edge[maxm];
int head[4], cnt;
long long d[4][maxn], w;

void add_edge (int u, int v, long long w) {
    edge[cnt] = (node) {u, v, head[u], w};
    head[u] = cnt++;
}
```

```
void solve () {
    priority_queue <pli, vector <pli>, greater<pli> > q;
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < w; j++) {
            d[i][j] = INF;
        }
    }
    d[1][0] = 0;
    q.push (make_pair (0, 1));
    while (!q.empty ()) {
        int u = q.top ().se;
        long long tmp = q.top ().fi;
        q.pop ();
        //cout << u << ".." << endl;
        if (tmp > d[u][tmp%w]) continue;
        for (int i = head[u]; i+1; i = edge[i].next) {
            int v = edge[i].v;
            long long dis = tmp + edge[i].w;
            if (dis < d[v][dis%w]) {
                d[v][dis%w] = dis;
                q.push (make_pair (dis, v));
            }
        }
    }
    long long ans = INF;
    for (int i = 0; i < w; i++) {
        if (d[1][i] >= K) {
            get_min (ans, d[1][i]);
        }
        else {
            long long more = (K-d[1][i]);
            long long cnt = (long long)ceil (1.0*more/w);
            get_min (ans, cnt*w+d[1][i]);
        }
    }
    printf ("%lld\n", ans);
}

int main () {
    int T; cin >> T;
    while (T--) {
        Clear (head, -1); cnt = 0;
        scan (K);
        for (int i = 0; i < 4; i++) {
            scan (a[i]);
            add_edge (i, (i+1)%4, a[i]);
            add_edge ((i+1)%4, i, a[i]);
        }
        w = min (a[0], a[1])<<1;
        solve ();
    }
    return 0;
}
```

平面图

对偶图

`BZOJ 4541`:给定平面图，每次询问一个图中的多边形求面积平方和以及面积和。 平面图生成对偶图之后，求出一个生成树，相当于求生成树上的一些子树，如果树上某条边对应进入多边形内部，就加上子树的贡献；如果是走出多边形，就减去子树贡献。

```
//如果maxn是平面图的点数 maxm是平面图的边数
//那么对偶图最多有2×maxn个点 maxm条边
int n, m, k, cnt;
int tot, root; //对偶图的总点数（平面图总面数） 对偶图的根（平面图的无穷大面）
struct Point {
    long long x, y;
    Point (long long _x = 0, long long _y = 0) {
        x = _x, y = _y;
    }
    Point operator + (Point &a) {
        return Point (x+a.x, y+a.y);
    }
    Point operator - (Point &a) {
        return Point (x-a.x, y-a.y);
    }
    long long operator * (Point a) { //有向面积
        return x*a.y-y*a.x;
    }
}p[maxn];

struct E {
    int u, v, id;//平面图每条边连接的两个点编号 这条边的编号
    double ang; //这条边的极角
    E (int _u = 0, int _v = 0, int _id = 0) {
        u = _u, v = _v, id = _id; ang = atan2 (double (p[u].y-p[v].y), double
(p[u].x-p[v].x));
    }
    bool operator < (const E &a) const {
        return ang < a.ang;
    }
}e[maxm<<1];
vector <E> Edge[maxn];

struct EE { //用来存对偶图
    int u, v, next, id; //id表示对应原图上的边
}edge[maxm<<1];
int tol, head[maxn<<1];
void add_edge (int u, int v, int id) {
    edge[tol] = (EE) {u, v, head[u], id}; head[u] = tol++;
```

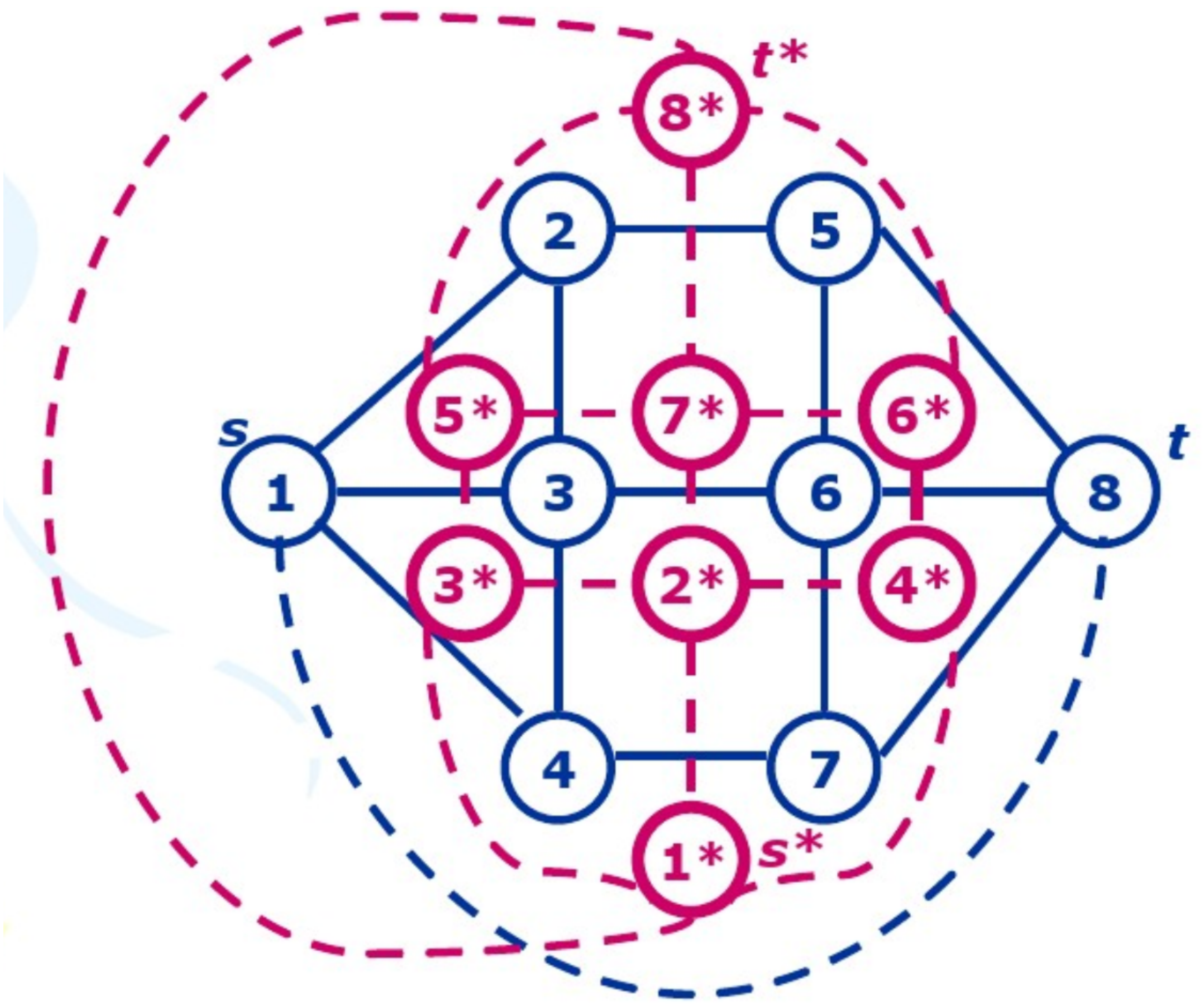
```
}
int next[maxm<<1]; //每条边的对应反边极角最接近的那条边
int num[maxm<<1]; //每条边属于哪一个点
long long S[maxn<<1]; //每个面（对偶图上的点）对应的面积的两倍 无穷大面是负数
void build_graph (int n, int m) { //读取一个n个点m条边的平面图 并转化成对偶图
    cnt = -1;
    for (int i = 1; i <= n; i++) {
        scanf ("%lld%lld", &p[i].x, &p[i].y);
        Edge[i].clear ();
    }
    for (int i = 1; i <= m; i++) {
        int u, v;        scanf ("%d%d", &u, &v);
        cnt++; e[cnt] = E (u, v, cnt); Edge[u].pb (e[cnt]);
        cnt++; e[cnt] = E (v, u, cnt); Edge[v].pb (e[cnt]);
    }
    for (int i = 1; i <= n; i++) sort (ALL (Edge[i]));
    for (int i = 0; i <= cnt; i++) {
        E &cur = e[i];
        int tmp = lower_bound (ALL (Edge[cur.v]), e[i^1])-Edge[cur.v].begin ()-1;
        if (tmp < 0) tmp += Edge[cur.v].size ();
        next[i] = Edge[cur.v][tmp].id;
    }
    //转化成对偶图
    int now, s; long long area; Clear (num, 0); tot = 0;
    for (int i = 0; i <= cnt; i++) if (!num[i]) {
        area = 0;
        now = i; s = e[i].u; num[i] = ++tot;
        while (1) {
            int tmp = next[now]; num[tmp] = tot;
            if (e[tmp].v == s) break;
            area += (p[e[tmp].u]-p[s])*(p[e[tmp].v]-p[s]);
            now = tmp;
        }
        S[tot] = area;
        if (area <= 0) root = tot; //无穷大的那个面
    }
    Clear (head, -1); tol = 0;
    for (int i = 0; i <= cnt; i++) {
        add_edge (num[i], num[i^1], i);
    }
}

long long sz[maxn<<1], sum[maxn<<1]; bool vis[maxn<<1], mark[maxm<<1];
int fa[maxn<<1];
void dfs (int u) {
    sz[u] = S[u]; vis[u] = 1; sum[u] = 1LL*S[u]*S[u];
    for (int i = head[u]; i+1; i = edge[i].next) {
        int v = edge[i].v; if (vis[v]) continue;
        mark[edge[i].id] = mark[edge[i].id^1] = 1;
        dfs (v);
        sz[u] += sz[v];
        sum[u] += sum[v];
        fa[v] = u;
    }
}

int tmp[maxn];
void solve () {
    Clear (vis, 0); Clear (mark, 0);
    fa[root] = 0; dfs (root);
    long long a = 0, b = 0; int d;
    while (k--) {
        scanf ("%d", &d);
        d = (d+a)%n+1;
        for (int i = 1; i <= d; i++) {
            scanf ("%d", &tmp[i]);
            tmp[i] = (tmp[i]+a)%n+1;
        }
        a = b = 0; tmp[d+1] = tmp[1];
        for (int i = 1; i <= d; i++) {
            int u = tmp[i], v = tmp[i+1], id = lower_bound (ALL (Edge[u]), E (u, v,
0))-Edge[u].begin ();
            id = Edge[u][id].id;
            if (!mark[id]) continue; //这条边不是树上的边
            if (num[id] == fa[num[id^1]]) a += sum[num[id^1]], b += sz[num[id^1]];
            else a -= sum[num[id]], b -= sz[num[id]];
        }
        if (b < 0) a *= -1, b *= -1;
        long long g = __gcd (a, b); a /= g, b /= g;
        if (a&1) b <<= 1; else a >>= 1;
        printf ("%lld %lld\n", a, b);
    }
}

int main () {
    scanf ("%d%d%d", &n, &m, &k);
    build_graph (n, m);
    solve ();
    return 0;
}
```

一类平面图上最小割



像这样的平面图s-t最小割，可以给s和t连边，然后构造对偶图和新源汇，在对偶图上跑新源点的最短路即为最小割。

连通性及缩点

求割点

求桥

```
struct node {
    int u, v, next;
    bool is;//这条边是不是桥
}edge[maxm];
int head[maxn], cnt;
int n, m;
int dfs_clock, low[maxn], pre[maxn];

void add_edge (int u, int v) {
    edge[cnt].is = 0;
    edge[cnt].u = u, edge[cnt].v = v, edge[cnt].next = head[u], head[u] = cnt++;
}
```

```
void find_cut (int u, int father) {
    pre[u] = low[u] = ++dfs_clock;
    for (int i = head[u]; i+1; i = edge[i].next) {
        int v = edge[i].v;
        if (v == father) continue;
        if (!pre[v]) {
            find_cut (v, u);
            if (low[v] <= pre[u]) {}
            else {
                edge[i].is = edge[i^1].is = 1;
            }
            low[u] = min (low[u], low[v]);
        }
        else {
            low[u] = min (low[u], pre[v]);
        }
    }
}

int main () {
    ios::sync_with_stdio (0);
    cin >> n >> m;
    memset (head, -1, sizeof head);
    cnt = 0;
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        add_edge (u, v);
        add_edge (v, u);
    }
    dfs_clock = 0;
    for (int i = 1; i <= n; i++) if (!pre[i])
        find_cut (i, 0);
    return 0;
}
```

点双联通分量

```
int n, m, k;
int pre[maxn],dfs_clock,iscut[maxn],low[maxn],bcc_cnt,bccno[maxn];
//iscut: 是否是割点 bcc_cnt: 点双联通分量个数 bccno: 每个点属于的分量
vector<int> bcc[maxn];
//bcc: 每个点双联通分量中的点
struct Edge {
    int u, v, w, next;
}edge[maxm];
int head[maxn], cnt;

void add_edge (int u, int v, int w) {
    edge[cnt] = (Edge) {u, v, w, head[u]}; head[u] = cnt++;
}

stack <Edge> S;
int tarjan (int u,int fa) {
    int lowu = pre[u] = ++dfs_clock;
    int child = 0;
    for (int i = head[u]; i+1; i = edge[i].next) {
        int v = edge[i].v;
        Edge e = (Edge) {u, v, 0, 0};
        if (!pre[v]) {
            S.push(e);
            child++;
            int lowv = tarjan (v,u);
            lowu = min(lowv,lowu);
            if (lowv >= pre[u]) {
                iscut[u] = true;
                bcc_cnt++; bcc[bcc_cnt].clear();
                for (;;) {
                    Edge x = S.top();S.pop();
                    if (bccno[x.u] != bcc_cnt) {
                        bcc[bcc_cnt].push_back (x.u);
                        bccno[x.u] = bcc_cnt;
                    }
                    if (bccno[x.v] != bcc_cnt) {
                        bcc[bcc_cnt].push_back (x.v);
                        bccno[x.v] = bcc_cnt;
                    }
                    if (x.u == u && x.v == v) break;
                }
            }
        }
        else if (pre[v] < pre[u] && v != fa) {
            S.push (e);
            lowu = min (lowu, pre[v]);
        }
    }
    if (fa < 0 && child == 1) iscut[u]=0;
    return lowu;
}

void find_bcc () {
    memset (pre,0,sizeof(pre));
    memset (iscut,0,sizeof(iscut));
    memset (bccno,0,sizeof(bccno));
    dfs_clock = bcc_cnt = 0;
    for (int i = 1; i <= n; i++)
        if (!pre[i]) tarjan (i,-1);
}
```

边双联通分量

```cpp
//已经处理了重边问题
#define index Index
struct node {
    int u, v, next;
}edge[maxm];
int head[maxn], cnt;
int low[maxn], dfn[maxn], Stack[maxn], belong[maxn];
int index, top;
int block; //边双联通分量的数量
bool vis[maxn];

int n, m;

void add_edge (int u, int v) {
    edge[cnt] = (node) {u, v, head[u]}; head[u] = cnt++;
}

void tarjan (int u, int pre) {
    int v;
    low[u] = dfn[u] = ++index;
    Stack[top++] = u;
    vis[u] = 1;
    for (int i = head[u]; i+1; i = edge[i].next) {
        v = edge[i].v;
        if (pre+1 && pre == (i^1)) continue;
        if (!dfn[v]) {
            tarjan (v, i);
            if (low[u] > low[v]) low[u] = low[v];
        }
        else if (vis[v] && low[u] > dfn[v]) {
            low[u] = dfn[v];
        }
    }
    if (low[u] == dfn[u]) {
        block++;
        do {
            v = Stack[--top];
            vis[v] = 0;
            belong[v] = block;
        }
        while (v != u);
    }
}

void find_bcc () {
    Clear (dfn, 0);
    Clear (vis, 0);
    index = top = block = 0;
    tarjan (1, -1);
}
```

有向图强连通分量

```
int dfn[maxn], low[maxn];
int sccno[maxn], scc_cnt, dfs_clock;
stack <int> S;

void tarjan (int u) {
    dfn[u] = low[u] = ++dfs_clock;
    S.push (u);
    for (int i = head[u]; i+1; i = edge[i].next) {
        int v = edge[i].v;
        if (!dfn[v]) {
            tarjan (v);
            low[u] = min (low[v], low[u]);
        }
        else if (!sccno[v]) {
            low[u] = min (low[u], dfn[v]);
        }
    }
    int now = 0;
    if (dfn[u] == low[u]) {
        scc_cnt++;
        for (; ;) {
            int x = S.top (); S.pop ();
            sccno[x] = scc_cnt;
            if (x == u) break;
        }
    }
}

void find_scc () {
    Clear (sccno, 0);
    Clear (dfn, 0);
    dfs_clock = scc_cnt = 0;
    for (int i = 1; i <= n; i++) if (!dfn[i])
        tarjan (i);
}
```

拓扑排序

2-sat

给出n种二元关系，根据"选择xx就必须选xx"建图

```
int mark[maxn], sta[maxn], top;
int dfs(int x)
{
    if(mark[x^1])
        return 0;
    if(mark[x])
```

```
        return 1;
    mark[x] = 1;
    sta[top++] = x;
    for(int i = head[x]; i+1; i = edge[i].next) {
        int v = edge[i].v;
        if (!dfs(v))
            return 0;
    }
    return 1;
}
int solve()
{
    top = 0;
    Clear (mark, 0);
    for(int i = 0; i < 2*m; i += 2)
    {
        if(!mark[i] && !mark[i+1])
        {
            top = 0;
            if(!dfs(i))
            {
                while(top > 0)
                    mark[sta[--top]] = 0;
                if(!dfs(i+1))
                    return 0;
            }
        }
    }
    return 1;
}
```

二分图

二分图的一些结论

最大独立集：最大的点集使得任意两点之间没有边 最小定点覆盖：最小的点集使得任意一个边和集合中一个点关联 $二分图最大匹配 = 节点总数 - 最大独立集 = 最小点覆盖$
$二分图最大点权独立集 = 总权值 - 二分图最小点权覆盖 = 总权值 - 最大流$

- 最小点覆盖的构造： 设二分点集是X,Y，每次从X中某个非匹配点出发，按照非匹配边-匹配边-非匹配边-匹配边...的路径增广，同时标记路径上的点，X中的非标记点和Y中的标记点构成一个最小点覆盖。
- 最大独立集的构造： 取整个点集的最小点覆盖的补集就是最大独立集。

####二分图最大匹配

- 匈牙利算法，复杂度$O(nm)$

```
int pre[maxn], vis[maxn];
bool dfs (int u) {
    for (int i = head[u]; i != -1; i = edge[i].next) {
        int v = edge[i].v;
```

```
            if (!vis[v]) {
                vis[v] = 1;
                if (pre[v] == -1 || dfs (pre[v])) {
                    pre[v] = u;
                    return 1;
                }
            }
        }
        return 0;
}

int hungry () {
    int ans = 0;
    memset (pre, -1, sizeof pre);
    for (int i = 1; i <= n; i++) {
        memset (vis, 0, sizeof vis);
        if (dfs (i))
            ans++;
    }
    return ans;
}
```

- HK算法，复杂度$O(\sqrt{n}m)$

```
//左右两类点的最大点数
#define maxL 100005
#define maxR 100005
int Mx[maxL], My[maxR], dx[maxL], dy[maxR]; bool used[maxR];
int dis;

bool searchP () {
    queue <int> q;
    dis = 1e9;
    Clear (dx, -1); Clear (dy, -1);
    for (int i = 0; i < n; i++) {
        if (Mx[i] == -1) {
            q.push (i);
            dx[i] = 0;
        }
    }
    while (!q.empty ()) {
        int u = q.front (); q.pop ();
        if (dx[u] > dis) break;
        for (int i = head[u]; i+1; i = edge[i].next) {
            int v = edge[i].v;
            if (dy[v] == -1) {
                dy[v] = dx[u]+1;
                if (My[v] == -1) dis = dy[v];
                else {
                    dx[My[v]] = dy[v]+1;
                    q.push (My[v]);
                }
            }
        }
    }
    return dis != 1e9;
}

bool dfs (int u) {
    for (int i = head[u]; i+1; i = edge[i].next) {
        int v = edge[i].v;
        if (!used[v] && dy[v] == dx[u]+1) {
            used[v] = 1;
            if (My[v] != -1 && dy[v] == dis) continue;
            if (My[v] == -1 || dfs (My[v])) {
                My[v] = u;
                Mx[u] = v;
                return 1;
            }
        }
    }
    return 0;
}

int maxflow () {
    int res = 0;
    Clear (Mx, -1);
    Clear (My, -1);
    while (searchP ()) {
        Clear (used, 0);
        for (int i = 0; i < n; i++) if (Mx[i] == -1 && dfs (i)) //注意这里修改点数
            res++;
    }
    return res;
}
```

二分图最大权匹配

```
//用邻接矩阵存图
int w[maxn][maxn];
int lx[maxn], ly[maxn], left[maxn];
bool s[maxn], t[maxn];

bool match (int i) {
    s[i] = 1;
    for (int j = 1; j <= n; j++) if (lx[i]+ly[j] == w[i][j] && !t[j]) {
        t[j] = 1;
        if (!left[j] || match (left[j])) {
            left[j] = i;
            return 1;
        }
    }
    return 0;
}

void update () {
    int a = INF;
    for (int i = 1; i <= n; i++) if (s[i])
        for (int j = 1; j <= n; j++) if (!t[j])
            a = min (a, lx[i]+ly[j]-w[i][j]);
    for (int i = 1; i <= n; i++) {
        if (s[i]) lx[i] -= a;
        if (t[i]) ly[i] += a;
    }
}

int km () {
    for (int i = 1; i <= n; i++) {
        left[i] = lx[i] = ly[i] = 0;
        for (int j = 1; j <= n; j++)
            lx[i] = max (lx[i], w[i][j]);
    }
    for (int i = 1; i <= n; i++) {
        for (; ;) {
            for (int j = 1; j <= n; j++) s[j] = t[j] = 0;
                if (match (i))
                    break;
                else update ();
        }
    }
    int ans = 0;
    for (int i = 1; i <= n; i++) {
        if (left[i]) {
            ans += w[left[i]][i];
        }
    }
    return ans;
}
```

最小路径覆盖

不可重复最小路径覆盖：求最少的不可重复路径数使得每个点都含在一条路径中。

`题意`：给出一个DAG，求最小路径覆盖以及输出路径。

- 最小路径覆盖 ＝ 总点数 － 最大匹配。输出路径按照pre数组输出即可。

```
int pre[maxn];
bool vis[maxn];
bool dfs (int u) {
    for (int i = head[u]; i != -1; i = edge[i].next) {
        int v = edge[i].v;
        if (!vis[v]) {
            vis[v] = 1;
            if (pre[v] == -1 || dfs (pre[v])) {
                pre[v] = u;
                return 1;
            }
        }
    }
    return 0;
}

int hungry () {
    int ans = 0;
    memset (pre, -1, sizeof pre);
    for (int i = 1; i <= n; i++) {
        memset (vis, 0, sizeof vis);
        if (dfs (i)) {
            ans++;
        }
    }
    return ans;
}

vector <int> path;

void go (int u) {
    vis[u] = 1;
    path.push_back (u);
    for (int v = 1; v <= n; v++) if (!vis[v] && pre[v] == u) {
        go (v);
        return ;
    }
}

void solve () {
    int ans = hungry ();
    memset (vis, 0, sizeof vis);
    for (int i = 1; i <= n; i++) {
        if (!vis[i] && pre[i] == -1) {
            path.clear ();
            go (i);
            for (int j = 0; j < path.size (); j++) {
                printf ("%d%c", path[j], (j == path.size()-1 ? '\n' : ' '));
            }
        }
    }
    cout << n-ans << endl;
}

int main () {
    while (cin >> n >> m) {
        memset (head, -1, sizeof head);
        cnt = 0;
        for (int i = 0; i < m; i++) {
            int u, v;
            cin >> u >> v;
            add_edge (u, v);
        }
        solve ();
    }
    return 0;
}
```

可重复的最小路径覆盖：用floyd求出传递闭包后转化成不可重复最小路径覆盖问题。

`BZOJ 1143`:

```
```

网络流

最大流&&最小割

- 用#define type xxx替代流的类型(int, double, long long)
- **极其关键：边数开两倍**

```
int n;
int s, t;
struct Edge {
    int from, to, next;
    type cap, flow;
    void get (int u, int a, int b, type c, type d) {
        from = u; to = a; next = b; cap = c; flow = d;
    }
}edge[maxm];
int tol;
int head[maxn];
int gap[maxn], dep[maxn], pre[maxn], cur[maxn];
void init () {
    tol = 0;
    memset (head, -1, sizeof head);
}

void add_edge (int u, int v, type w, type rw=0) {
    edge[tol].get(u, v,head[u],w,0);head[u]=tol++;
    edge[tol].get(v, u,head[v],rw,0);head[v]=tol++;
}

type sap (int start, int end, int N) {
    memset (gap, 0, sizeof gap);
    memset (dep, 0, sizeof dep);
    memcpy (cur, head, sizeof head);
    int u = start;
    pre[u] = -1;
    gap[0] = N;
    type ans = 0;
    while (dep[start] < N) {
        if (u == end) {
            type Min = INF;
            for (int i = pre[u]; i != -1; i = pre[edge[i^1].to])
                if (Min > edge[i].cap - edge[i].flow)
                   Min = edge[i].cap-edge[i].flow;
            for (int i = pre[u]; i != -1; i = pre[edge[i^1].to]) {
                edge[i].flow += Min;
                edge[i^1].flow -= Min;
            }
            u = start;
            ans += Min;
            continue;
        }
        bool flag = false;
        int v;
        for (int i = cur[u]; i != -1; i = edge[i].next) {
            v = edge[i].to;
            if (edge[i].cap - edge[i].flow && dep[v]+1 == dep[u]) {
                flag = true;
                cur[u] = pre[v] = i;
                break;
            }
        }
        if (flag) {
            u = v;
            continue;
        }
        int Min = N;
        for (int i = head[u]; i != -1; i = edge[i].next)
            if (edge[i].cap - edge[i].flow && dep[edge[i].to] < Min) {
                  Min = dep[edge[i].to];
                  cur[u] = i;
             }
        gap[dep[u]]--;
        if (!gap[dep[u]]) return ans;
        dep[u] = Min+1;
        gap[dep[u]]++;
        if (u != start) u = edge[pre[u]^1].to;
    }
    return ans;
}
```

- 最小割的可行边/必须边[BZOJ 1797]

跑最大流之后的残余网络进行强连通缩点，对于**满流**边<u,v>： 可行边：当且仅当id[u]!=id[v]； 必须边：当且仅当id[u]id[S]&&id[v]id[T]。

- 一类最小割建图

将所有的收益/花费分成两类，问最大的收益。 `BZOJ 3511/3438/2039` ：将图上的点分成两类，一类和点1在一起，另一类和点n在一起。每个点属于第一类的收益是x，属于第二类的收益是y；每条边两个端点都属于第一类的收益是a，都属于第二类的收益是b，否则倒扣c。问最大收益的划分方式。

构图方式： 将和S相连看成分在第一类，和T相连分在第二类。那么如果一个点分在第一类就要放弃第二类的收益（连点到T，流量为y），否则就要放弃第一类收益（连S到点，流量为x）；对于一种边，要么全部在第一类，放弃第二类收益（连两个端点到T，流量为b/2），要么来全在第二类（连S到两个点，流量为a/2），**要么分开归类（端点互连，流量为a/2+b/2+c）**，这样的用意是如果存在边连着S和T，那么必须要把这个边中的代价计入（即倒扣c）。最后的答案=总收益-最小割。

- 最大权闭合图

闭合图:子图中任意点的所有后继都在子图中。

最大权闭合图:点权和最大的闭合图。

建模:

1. 原图的边，流量为INF；
2. S->正权点，流量为abs(权值)；
3. 负权点->T，流量为abs(权值)；
4. 最大权=正权点之和-最小割；
5. 残留网络中S出发能够dfs的点都是最大权闭合图中的点。

- 最大密度子图 求最大的子图使得:边数/点数最大。
- S->T经过一个特定点，每个点只能经过一次的路径（如果要最短加个费用即可）。

1. 超级源点->mid，流量为2；
2. 其他所有点拆点流量为1；
3. 其他所有的边流量为1；
4. S，T->超级汇点，流量为1。

`2015北京D` :给出n个技能，m个关系（学某个技能前必须先学会某一个技能）。可以直接通过氪金花费一定的钱学习一个技能（不需要考虑任何关系），可以花费一定的钱删掉某一种关系，以及正常学一个技能需要花费一定的钱。问学习目标技能最少花费多少。 把每个点i拆成i+n：

1. S到i，流量为正常花费；
2. i到i+n，流量为氪金；
3. 每一个关系$<u,v,w>$，u+n到v，流量为w；
4. 目标+n到T，流量为INF。 最小割就是最小花费。

费用流

```
int n, m;
int s, t;
struct node {
    int u, v, next;
    type cap, flow, cost;
}edge[maxm];
```

```
int head[maxn], cnt;
int pre[maxn];
type dis[maxn];
bool vis[maxn];
int N;

void init () {
    memset (head, -1, sizeof head);
    cnt = 0;
}

void add_edge (int u, int v, type cap, type cost) {
    edge[cnt] = (node) {u, v, head[u], cap, 0, cost}; head[u] = cnt++;
    edge[cnt] = (node) {v, u, head[v], 0, 0, -cost}; head[v] = cnt++;
}

bool spfa (int s, int t) {
    queue <int> q;
    for (int i = 0; i < N; i++) {
        dis[i] = INF;
        vis[i] = 0;
        pre[i] = -1;
    }
    dis[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 != -1; i = edge[i].next) {
            int v = edge[i].v;
            if (edge[i].cap > edge[i].flow && dis[v] > dis[u]+edge[i].cost) {
                dis[v] = dis[u]+edge[i].cost;
                pre[v] = i;
                if (!vis[v]) {
                    vis[v] = 1;
                    q.push (v);
                }
            }
        }
    }
    if (pre[t] == -1)
        return 0;
    else
        return 1;
}

int MCMF (int s, int t, type &cost) {
    type flow = 0;
    cost = 0;
    while (spfa (s, t)) {
        type Min = INF;
        for (int i = pre[t]; i != -1; i = pre[edge[i^1].v]) {
            if (Min > edge[i].cap-edge[i].flow) {
                Min = edge[i].cap-edge[i].flow;
            }
        }
        for (int i = pre[t]; i != -1; i = pre[edge[i^1].v]) {
            edge[i].flow += Min;
            edge[i^1].flow -= Min;
            cost += edge[i].cost*Min;
        }
        flow += Min;
    }
    return flow;
}
```

`2015沈阳L`:给出一个网格图，要把奇数和偶数连起来，然后所有剩下的空格子连城若干个环(两个点可以重复走构成一个环)。已知每一对格子之间连边的花费，求最小费用。 把每一个格子拆成入点和出点，然后这样建模：

1. S到奇数入点建边，流量为1费用为0；
2. 偶数出点到T建边，流量为1费用为0；
3. S到空点入点建边，空点出点到T建边，流量为1费用为0；
4. 每一个点的入点到上下左右的点的出点建边，流量为1费用为连边费用。 这样跑一次费用流判断是否满流即可。因为这样的最大流必然是空格子参与某一条奇偶路径或者环。

```
int get (int x, int y) {
    return x*m+y;
}

int getint () {
    bool s=0;
    int a,f;
    while(a=getchar(),!(a>='0'&&a<='9')&&a!='-');
    if(a-'-')
        a-='0';
    else
        a=0,s=1;
    while(f=getchar(),f>='0'&&f<='9')
        a=a*10+f-'0';
    return s ? -a:a;
}

int main () {
    int tt, kase = 0, maxflow;
    scanf ("%d", &tt);
    while (tt--) {
        printf ("Case #%d: ", ++kase);
        n = getint (), m = getint ();
        maxflow = 0;
        init ();
        N = 2*n*m+2, s = N-2, t = N-1;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                int num;
                num = getint ();
                if (!num) {
                    add_edge (s, get (i, j), 1, 0);
                    add_edge (get (i, j)+n*m, t, 1, 0);
                    maxflow++;
                }
                else if (num&1) {
                    add_edge (s, get (i, j), 1, 0);
                    maxflow++;
                }
                else {
                    add_edge (get (i, j)+n*m, t, 1, 0);
                }
            }
        }
        for (int i = 0; i < n-1; i++) {
            for (int j = 0; j < m; j++) {
                int num; num = getint ();
                add_edge (get (i, j), get (i+1, j)+n*m, 1, num);
                add_edge (get (i+1, j), get (i, j)+n*m, 1, num);
            }
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m-1; j++) {
                int num; num = getint ();
                add_edge (get (i, j), get (i, j+1)+n*m, 1, num);
                add_edge (get (i, j+1), get (i, j)+n*m, 1, num);
            }
        }
        long long cost;
        int res;
        res = MCMF (s, t, cost);
        if (res != maxflow) {
            puts ("-1");
        }
        else
            printf ("%lld\n", cost);
    }
    return 0;
}
```

有上下界网络流

- 无源汇可行流

假设原始边$< u, v, l, r >$表示u到v的一条容量范围是$[l, r]$的边，增加超级源汇S和T并把它替换成：

1. u到T容量为l的边；
2. S到v容量为l的边；
3. u到v容量为$r-l$的边； 跑S到T的最大流，存在可行流当且仅当所有S出发的边满流。 可行方案：在残余网络上对每条边$<u,v>.flow+l$即可。

`SGU 194`:输出无源汇可行流方案。

```
int l[maxm], r[maxm];
void solve () {
    for (int i = 0; i < m; i++) {
        printf ("%d\n", l[i]+edge[i*6+4].flow);
    }
}

int main () {
    init ();
    cin >> n >> m;
    s = 0, t = n+1; int N = t+1, sum = 0;
    for (int i = 0; i < m; i++) {
        int u, v; cin >> u >> v >> l[i] >> r[i];
        add_edge (s, v, l[i]); add_edge (u, t, l[i]); add_edge (u, v, r[i]-l[i]);
        sum += l[i];
    }
    int flow = sap (s, t, N);
    if (flow >= sum) {
        printf ("YES\n");
        solve ();
    }
    else {
        puts ("NO");
    }
    return 0;
}
```

- 有源汇最大流

设初始源汇点是s和t，增加超级源汇点S和T，连边$<t,s,INF>$转化成了无源汇图。首先按照无源汇方式判定跑一遍$sap(S,T)$判断是否存在可行流，然后再跑一遍$sap(s,t)$得到最大可行流。

- 有源汇最小流

设初始源汇点是s和t，增加超级源汇点S和T，

1. 跑$sap(S,T)$后连边$<t,s$，$INF>$，设这条边的编号是id，
2. 跑$sap(S,T)$，若此时所有S出发的边都满流则存在可行解
3. 最小流是$edge[id\ xor\ 1].flow$，求方案和无源汇的方案求法一致。

`SGU 176`:给出一个网络，每条边$<u,v,w,op>$表示u到v容量上限是w，如果op为0下限是w，否则是0.求一个最小流。

```
int du[maxn], ans[maxm]; map<pii, int> mp; //每条边的编号
int main () {
    Clear (du, 0); init (); mp.clear (); Clear (ans, 0);
    cin >> n >> m;
    int s = 1, t = n, S = 0, T = n+1, N = T+1;
    for (int i = 0; i < m; i++) {
        int u, v, w; int op;
        cin >> u >> v >> w >> op;
        if (op == 0) {
            add_edge (u, v, w); ans[i] = 0;
        }
        else {
            du[v] += w; du[u] -= w; ans[i] = w;
        }
        mp[make_pair (u, v)] = i;
    }
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        if (du[i] > 0) {
            add_edge (S, i, du[i]); sum += du[i];
        }
        if (du[i] < 0) {
            add_edge (i, T, -du[i]);
        }
    }
    int flow = sap (S, T, N);
    sap (S, T, N);
    int id; add_edge (t, s, INF);
    sap (S, T, N);
    for (int i = head[S]; i+1; i = edge[i].next) {
        if (edge[i].flow < edge[i].cap) {
            puts ("Impossible"); return 0;
        }
    }
    for (int i = head[t]; i+1; i = edge[i].next) {
        if (edge[i].to == s) {id = i^1; break;}
    }
    cout << edge[id^1].flow << endl;
    for (int u = 1; u <= n; u++) {
        for (int i = head[u]; i+1; i = edge[i].next) if (i%2 == 0) { //保证是个正向边
            int v = edge[i].to; if (v < 1 || v > n || (u == t && v == s)) continue;
            int tmp = mp[make_pair (u, v)];
            ans[tmp] += edge[i].flow;
        }
    }
    for (int i = 0; i < m; i++) printf ("%d%c", ans[i], i == m-1 ? '\n' : ' ');
    return 0;
}
```

其他图论

弦图

弦：无向图中环上不相邻的点之间的边 弦图：对于图上任何一个$n(n \geq 3)$元环，都存在弦

- 弦图判定：最大势算法（BZOJ 1242）

```
int n, m;
bool c[maxn][maxn];
int qu[maxn], inq[maxn], dgr[maxn];
int stk[maxn], top;

void msc() {
    dgr[0] = -1;
    for( int i=n; i>=1; i-- ) {
        int s = 0;
        for( int u=1; u<=n; u++ )
            if( !inq[u] && dgr[u]>dgr[s] ) s=u;
        qu[i] = s;
        inq[s] = true;
        for( int u=1; u<=n; u++ )
            if( !inq[u] && c[s][u] ) dgr[u]++;
    }
}
bool check() {
    for( int i=n; i>=1; i-- ) {
        int s=qu[i];
        top = 0;
        for( int j=i+1; j<=n; j++ )
            if( c[s][qu[j]] ) stk[++top] = qu[j];
        if( top==0 ) continue;
        for( int j=2; j<=top; j++ )
            if( !c[stk[1]][stk[j]] ) return false;
    }
    return true;
}
int main() {
    scanf( "%d%d", &n, &m );
    for( int i=1,u,v; i<=m; i++ ) {
        scanf( "%d%d", &u, &v );
        c[u][v] = c[v][u] = 1;
    }
    msc();
    printf( "%s\n", check() ? "Perfect" : "Imperfect" );
}
```

- 弦图的最小染色数（BZOJ 1006）

```
bool vis[maxn];
int d[maxn], q[maxn], col[maxn], hash[maxn];
int solve () {
//弦图染色的最小色数
    int ans = 0;
    for (int i = n; i; i--) {
        int t = 0;
        for (int j = 1; j <= n; j++) {
            if (!vis[j] && d[j] >= d[t]) t = j;
        }
        vis[t] = 1; q[i] = t;
        for (int j = head[t]; j+1; j = edge[j].next) d[edge[j].v]++;
    }
    for (int i = n; i; i--) {
        int t = q[i];
        for (int j = head[t]; j+1; j = edge[j].next) hash[col[edge[j].v]] = i;
        int j;
        for (j = 1; ; j++) if (hash[j] != i) break;
        col[t] = j;
        if (j > ans) ans = j;
    }
    return ans;
}
```

欧拉路径

- 有向图欧拉通路

`POJ 2337`:给出n个单词，求出最小字典序的头尾连接方案。 把每个单词当做边，头字母和尾字母当做节点，建完跑欧拉通路即可。

```
int n;
string str[maxn];
struct node {
    int v;
    string s;
    bool vis;
};
int in[maxn], out[maxn], cnt, from[maxn];
vector <node> g[maxn];
vector <string> ans;

void dfs (int u) {
    int sz = g[u].size ();
    for (int i = from[u]; i < sz; i++) {//from数组用于剪枝
        if (!g[u][i].vis) {
            g[u][i].vis = 1;
            from[u] = i+1;
            dfs (g[u][i].v);
            ans.pb (g[u][i].s);
        }
        else break;//后面的必然已经搜过
    }
}

int main () {
```

```cpp
    Close ();
    int T; cin >> T;
    while (T--) {
        cin >> n;
        for (int i = 0; i < n; i++) cin >> str[i];
        sort (str, str+n);
        Clear (in, 0);
        Clear (out, 0);
        Clear (from, 0);
        int s = 233;
        for (int i = 0; i < 26; i++) g[i].clear ();
        for (int i = 0; i < n; i++) {
            int u = str[i][0]-'a', v = str[i][str[i].length()-1]-'a';
            g[u].push_back ((node) {v, str[i], 0});
            in[v]++;
            out[u]++;
            get_min (s, min (u, v));//先随便找个最小的起点
        }
        int x = 0, y = 0;//出度比入度多1的点(必是起点) 入度比出度多1的点
        bool ok = 1;
        for (int i = 0; i < 26; i++) {
            if (out[i]-in[i] == 1) {
                s = i;
                x++;
            }
            if (in[i]-out[i] == 1) y++;
            else if (abs (in[i]-out[i]) > 1) {
                ok = 0;
            }
        }
        if ((x^y) || !ok || x > 1 || y > 1) {
            //有欧拉回路的必要条件
            cout << "***\n";
            continue;
        }
        ans.clear ();
        dfs (s);
        int sz = ans.size ();
        if (sz != n) {
            cout << "***\n";
            continue;
        }
        for (int i = sz-1; i >= 0; i--) {
            cout << ans[i];
            if (i > 0) cout << ".";
            else cout << "\n";
        }
    }
    return 0;
}
```

prufer序列

prufer数列是一种无根树的编码表示，对于一棵n个节点带编号的无根树，对应唯一一串长度为n-2的prufer编码。

- 无根树转化为prufer序列。 首先定义无根树中度数为1的节点是叶子节点；找到编号最小的叶子并删除，序列中添加与之相连的节点编号，重复执行直到只剩下2个节点。
- prufer序列转化为无根树。 设点集$V=\{1,2,3\dots n\}$，每次取出prufer序列中最前面的元素u，在V中找到编号最小的没有在prufer序列中出现的元素v，给u，v连边然后分别删除，最后在V中剩下两个节点，给它们连边。最终得到的就是无根树。

有一个很重要的性质就是prufer序列中某个编号出现的次数+1就等于这个编号的节点在无根树中的度数。

数学

基本数论&&推导

一些结论

- $a\ XOR b + a\ AND b = a + b$
- $E(p_1A_1 + p_2A_2 + \dots + p_nA_n) = p_1E(A1) + p_2E(A_2) + \dots + p_nE(A_n)$
- B进制下一个如果一个数各个数位和是p的倍数那么这个数被p整除的充要条件是p是B-1的约数
- 能被7整除的数的性质：末三位之前的数和末三位的差能被7整除

关于gcd

- $gcd(x^a-1,x^b-1)=x^{gcd(a,b)}-1$
- $gcd(2a-1,2b-1)=1$当且仅当$gcd(a,b)=1$
- $gcd(a^n-b^n,a^m-b^m)=a^{gcd(n,m)}-b^{gcd(n,m)}$
- $gcd(fib_n,fib_m)=fib_{gcd(n,m)}$
- 长度为n的数组所有的$gcd(l,l+1\dots r)$只有$nlgn$种

一些求和公式

1. 平方和公式：$1^2+2^2+\dots+n^2=\frac{n(n+1)(n+2)}{6}$
2. 立方和公式：$1^3+2^3+\dots+n^3=(1+2+3+\dots+n)^2$
3. 错位加公式：$\sum_{i=1}^n i(n-i+1)=\frac{n(n+1)(n+2)}{6}$

求最大的模

- 最大化$a_i\%a_j$, 其中 $a_i>a_j$。

令$a_i=k\times a_j+b$,枚举a_j和k,每次找到数列中最后一个小于$k\times a_j$的数来更新答案就可以了.

求(0^n)+(1^n)+...+((n-1)^n)

按位计算贡献

```
long long solve (long long n) { //注意是否使用模数
    if (n == 0) return 0;
    int k = 0;
    while ((1LL<<(k+1)) <= n) k++; //n的最高位在哪里
    long long ans = 0, pre = 0, id;
    for (int i = k; i >= 0; i--, pre <<= 1, pre += id) { //按位算贡献
        long long tmp = 0;
        if (pre > 0) tmp = (tmp + (1LL<<i)%mod*(pre%mod)) %mod; //前面比他小 后面随意
        id = ((n&(1LL<<i))>0);
        if (id) tmp = (tmp + (1LL<<i)%mod) %mod; //前面和他相等 后面随意
        ans = (ans + (1LL<<i)%mod*tmp%mod) %mod;
    }
    return ans;
}
```

求B进制下n以内的回文数字

```
int numPalindrome(int num, int B)
{
    int numLength = 0;
    int palLength = 0;
    int palPrefix = 0;
    int temp      = 0;
    int i         = 0;

    for (numLength=0, temp = num; temp != 0; temp /= B)
        numLength++;
    palLength = (numLength+1) / 2;
    palPrefix = num;
    for (i=0; i < numLength - palLength; i++)
        palPrefix /= B;
    if (constructPalindrome(palPrefix, numLength, B) > num)
        palPrefix--;
    palPrefix *= 2;
    if (numLength & 1) {
        int adjustment = 1;
        for (i=1;i<palLength;i++)
            adjustment *= B;
        palPrefix -= (palPrefix/2 - adjustment + 1);
    } else {
        int adjustment = 1;
        for (i=0;i<palLength;i++)
            adjustment *= B;
        palPrefix += (adjustment - 1 - palPrefix/2);
    }
    return palPrefix;
}

int constructPalindrome(int palPrefix, int numLength, int B)
{
    int front = palPrefix;
    int back  = 0;
    if (numLength & 1)
        palPrefix /= B;
    while (palPrefix != 0) {
        int digit = palPrefix % B;

        palPrefix /= B;
        back       = back * B + digit;
        front     *= B;
    }
    return front + back;
}
```

快速乘法

```
long long mul (long long a,long long b,long long c) {
    a %= c;
    b %= c;
    long long ret = 0;
    long long tmp = a;
    while(b) {
        if(b & 1) {
            ret += tmp; if(ret > c)ret -= c;//直接取模慢很多
        }
        tmp <<= 1;
        if (tmp > c) tmp -= c; b >>= 1;
    }
    return ret;
}
```

快速幂

```
long long qpow (long long a, long long b, long long mod) {
    long long ret=1;
    while (b) {
        if (b&1) ret = (ret*a)%mod;
        a = (a*a)%mod;
        b >>= 1;
    }
    return ret;
}
```

逆元

求a模m意义下的逆元

- 欧拉函数,要求mod是质数,并且a和mod互质

```
qpow (a, mod-2, mod);
```

- 扩展欧几里得求逆元

```
long long mod_rev (long long a,long long n) {
    long long x,y;
    long long d = ex_gcd (a,n,x,y);
    if (d == 1) return (x%n+n)%n;
    else return -1;
}
```

- 逆元递推

```
long long rev[maxn];
void init () {
    f[1] = 1;
    for (int i = 2; i <= 1000000; i++){
        f[i] = (long long)(mod - mod/i) * f[mod % i] % mod;
    }
}
```

- 阶乘逆元打表

```
//fac数组存阶乘 rev数组存逆元
fac[0] = 1;
for (int i = 1; i < maxn; i++) fac[i] = fac[i - 1] * i % mod;
rev[maxn - 1] = qpow(fac[maxn - 1], mod - 2);
for (int i = maxn - 2; i >= 0; i--) rev[i] = rev[i + 1] * (i + 1) % mod;
```

[1,n]内素数个数

```
//可以做到1e11
#define MAXN 100
#define MAXM 50010
#define MAXP 666666
#define MAX 5000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define chkbit(ar, i) (((ar[(i) >> 6]) & (1 << (((i) >> 1) & 31))))
#define setbit(ar, i) (((ar[(i) >> 6]) |= (1 << (((i) >> 1) & 31))))
#define isprime(x) (( (x) && ((x)&1) && (!chkbit(ar, (x)))) || ((x) == 2))

namespace pcf{
    long long dp[MAXN][MAXM];
    unsigned int ar[(MAX >> 6) + 5] = {0};
    int len = 0, primes[MAXP], counter[MAX];

    void Sieve(){
        setbit(ar, 0), setbit(ar, 1);
        for (int i = 3; (i * i) < MAX; i++, i++){
            if (!chkbit(ar, i)){
                int k = i << 1;
                for (int j = (i * i); j < MAX; j += k) setbit(ar, j);
            }
        }

        for (int i = 1; i < MAX; i++){
            counter[i] = counter[i - 1];
            if (isprime(i)) primes[len++] = i, counter[i]++;
        }
    }

    void init(){
        Sieve();
        for (int n = 0; n < MAXN; n++){
            for (int m = 0; m < MAXM; m++){
                if (!n) dp[n][m] = m;
                else dp[n][m] = dp[n - 1][m] - dp[n - 1][m / primes[n - 1]];
            }
        }
    }

    long long phi(long long m, int n){
        if (n == 0) return m;
        if (primes[n - 1] >= m) return 1;
        if (m < MAXM && n < MAXN) return dp[n][m];
        return phi(m, n - 1) - phi(m / primes[n - 1], n - 1);
    }

    long long Lehmer(long long m){
        if (m < MAX) return counter[m];

        long long w, res = 0;
        int i, a, s, c, x, y;
        s = sqrt(0.9 + m), y = c = cbrt(0.9 + m);
        a = counter[y], res = phi(m, a) + a - 1;
        for (i = a; primes[i] <= s; i++) res = res - Lehmer(m / primes[i]) +
Lehmer(primes[i]) - 1;
        return res;
    }
}

int main () {
    pcf::init ();
    long long n;
    while (cin >> n) {
        cout << pcf::Lehmer(n) << endl;
    }
}
```

微积分

- 弧长积分公式：$ds = \sqrt{(1 + y'^2)}d_x$
- 面积积分公式：$S = \int f(x)d_x$

- 求积分：$\int_a^b f(x)d_x$: 自适应simpson积分,F表示积分函数，直接调用asr(a, b, eps). **注意精度的设置**，有时候需要在WA和TLE中调精度。

```
double F (double x) {
    return sqrt (b*b-b*b/a/a*x*x);
}

double simpson (double a, double b) {
    double c = a+(b-a)/2;
    return (F(a)+4*F(c)+F(b))*(b-a)/6;
}

double asr (double a, double b, double eps, double A) {
    double c = a+(b-a)/2;
    double L = simpson (a, c), R = simpson (c, b);
    if (fabs (L+R-A) <= 15*eps) return L+R+(L+R-A)/15.0;
    return asr (a, c, eps/2, L)+asr (c, b, eps/2, R);
}

double asr (double a, double b, double eps) {
    return asr (a, b, eps, simpson (a, b));
}
```

拉格朗日插值

`CF 622F`: 求$1^k + 2^k + 3^k \cdots + n^k$.

- 拉格朗日插值：k-1次多项式可以用k个点唯一确定，插值公式就是$F_k(x) = \sum_{i=1}^k y_i \times \prod_{j=1\&\&j!=i}^k \frac{x-x_j}{x_k-x_j}$

```
#define maxn 1000005
#define mod  1000000007

long long fac[maxn], y[maxn];
long long n, k;

long long qpow (long long a, long long b) {
    a %= mod, b %= mod;
    long long ans = 1;
    while (b) {
        if (b&1)
            ans = ans*a%mod;
        a = a*a%mod;
        b >>= 1;
    }
    return ans;
}

int main () {
    cin >> n >> k;
    fac[0] = 1;
    for (long long i = 1; i <= k+2; i++) {
        fac[i] = fac[i-1]*i%mod;
    }
    if (k == 0) {
        cout << n << endl;
        return 0;
    }
    y[0] = 0;
    for (long long i = 1; i <= k+2; i++) {
        y[i] = y[i-1]+qpow (i, k);
        y[i] %= mod;
    }
    if (n <= k+2) {
        cout << y[n] << endl;
        return 0;
    }
    long long ans1 = 1;
    for (long long i = n-k-2; i <= n-1; i++) ans1 = ans1*i%mod;
    long long ans = 0;
    for (long long i = 1; i <= k+2; i++) {
        long long p1 = qpow (n-i, mod-2)%mod;
        long long p2 = qpow (fac[i-1]*fac[k+2-i]%mod, mod-2)%mod;
        long long sign = ((k+2-i)%2 == 1 ? -1 : 1);
        long long cur = y[i]*p1%mod*p2%mod*ans1%mod;
        cur *= sign;
        if (cur < 0)
            cur = (cur+mod)%mod;
        ans = (ans+cur)%mod;
    }
    cout << ans << endl;
    return 0;
}
```

多项式

```
namespace polysum {
    #define rep(i,a,n) for (int i=a;i<n;i++)
    #define per(i,a,n) for (int i=n-1;i>=a;i--)
    const int D=101000;
    ll a[D],tmp[D],f[D],g[D],p[D],p1[D],p2[D],b[D],h[D][2],C[D];
    ll powmod(ll a,ll b){ll res=1;a%=mod;assert(b>=0);for(;b;b>>=1)
{if(b&1)res=res*a%mod;a=a*a%mod;}return res;}
    ll calcn(int d,ll *a,ll n) { // 预处理a[0].. a[d]  得到a[n]
        if (n<=d) return a[n];
        p1[0]=p2[0]=1;
        rep(i,0,d+1) {
            ll t=(n-i+mod)%mod;
            p1[i+1]=p1[i]*t%mod;
        }
        rep(i,0,d+1) {
            ll t=(n-d+i+mod)%mod;
            p2[i+1]=p2[i]*t%mod;
        }
        ll ans=0;
        rep(i,0,d+1) {
            ll t=g[i]*g[d-i]%mod*p1[i]%mod*p2[d-i]%mod*a[i]%mod;
            if ((d-i)&1) ans=(ans-t+mod)%mod;
            else ans=(ans+t)%mod;
        }
        return ans;
    }
    void init(int M) {//预处理阶乘&&阶乘的逆元
        f[0]=f[1]=g[0]=g[1]=1;
        rep(i,2,M+5) f[i]=f[i-1]*i%mod;
        g[M+4]=powmod(f[M+4],mod-2);
        per(i,1,M+4) g[i]=g[i+1]*(i+1)%mod;
    }
    ll polysum(ll n,ll *a,ll m) { // 预处理a[0].. a[m] 得到\sum_{i=0}^{n-1} a[i]
        rep(i,0,m+1) tmp[i]=a[i];
        tmp[m+1]=calcn(m,tmp,m+1);
        rep(i,1,m+2) tmp[i]=(tmp[i-1]+tmp[i])%mod;
        return calcn(m+1,tmp,n-1);
    }
    ll qpolysum(ll R,ll n,ll *a,ll m) { // a[0].. a[m] \sum_{i=0}^{n-1} a[i]*R^i
        if (R==1) return polysum(n,a,m);
        a[m+1]=calcn(m,a,m+1);
        ll r=powmod(R,mod-2),p3=0,p4=0,c,ans;
        h[0][0]=0;h[0][1]=1;
        rep(i,1,m+2) {
            h[i][0]=(h[i-1][0]+a[i-1])*r%mod;
            h[i][1]=h[i-1][1]*r%mod;
        }
        rep(i,0,m+2) {
            ll t=g[i]*g[m+1-i]%mod;
            if (i&1) p3=((p3-h[i][0]*t)%mod+mod)%mod,p4=((p4-h[i][1]*t)%mod+mod)%mod;
            else p3=(p3+h[i][0]*t)%mod,p4=(p4+h[i][1]*t)%mod;
        }
        c=powmod(p4,mod-2)*(mod-p3)%mod;
        rep(i,0,m+2) h[i][0]=(h[i][0]+h[i][1]*c)%mod;
        rep(i,0,m+2) C[i]=h[i][0];
        ans=(calcn(m,C,n)*powmod(R,n)-c)%mod;
        if (ans<0) ans+=mod;
        return ans;
    }
} // polysum::init();
```

扩展欧几里得

用于求解$ax+by=c$的一组最小解，有解当且仅当$c\%gcd(a,b)=0$。

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

- 合并同余：$x = a1(\%m1)$和$x = a2(\%m2)$.

```
pll merge (pll p, pll q) {
//合并同余方程x%m1=a1和x%m2=a2 返回对应的新的模数和余数 否则返回（-1, -1）
    long long m1 = p.fi, a1 = p.se, m2 = q.fi, a2 = q.se;
    pll ans = make_pair (-1, -1);
    if (m1 < 0) return ans;
    long long gcd, x, y; //m1 m2的gcd和一组x1m1-x2m2=gcd(m1,m2)的最小解
    gcd = ex_gcd (m1, m2, x, y);
    if (gcd == -1 || (a2-a1)%gcd != 0) return ans; //无解
    __int128 x0 = (__int128)((a2-a1)/gcd)*x % m2; //记得把等式右边还原成a2-a1 这是原方程一
个最小解
    //如果会爆longlong就要用__int128
    ans.fi = m1/gcd*m2;
    __int128 tmp = x0*m1+a1;
    tmp = (tmp%ans.fi+ans.fi)%ans.fi; ans.se = (long long)tmp;
    return ans;
}
```

大素数测试和大数分解

`POJ 1811`:<2^63的数字素数测试并求出最小的质因子。Miller_Rabin板子。

```
//Miller_Rabin 可以判断一个 < 2^63 的数是不是素数
const int S = 8; //随机算法判定次数，一般8~10就够了
// 计算 ret = (a^n)%mod
long long pow_mod(long long a,long long n,long long mod) {
    long long ret = 1;
    long long temp = a%mod;
    while (n) {
        if(n & 1)ret = mul (ret,temp,mod);
        temp = mul (temp,temp,mod);
        n >>= 1;
    }
    return ret;
}
// 通过 a^(n-1)=1(mod n)来判断n是不是素数
// n-1 = x*2^t 中间使用二次判断
// 是合数返回true, 不一定是合数返回false
bool check(long long a,long long n,long long x,long long t) {
    long long ret = pow_mod(a,x,n); long long last = ret;
    for(int i = 1;i <= t;i++)
    {
        ret = mul (ret,ret,n);
        if(ret == 1 && last != 1 && last != n-1)return true;//合数
        last = ret;
    }
    if(ret != 1)return true;
    else return false;
}
// Miller_Rabin算法
// 是素数返回true,(可能是伪素数),不是素数返回false
bool Miller_Rabin(long long n) {
    if( n < 2)return false;
    if( n == 2)return true;
    if( (n&1) == 0)return false;//偶数
    long long x = n - 1;
    long long t = 0;
    while( (x&1)==0 ){x >>= 1; t++;}
    srand((long long) time(NULL));
    for(int i = 0;i < S;i++) {
        long long a = rand()%(n-1) + 1;
        if( check(a,n,x,t) ) return false;
    }
    return true;
}
// pollard_rho 算法进行质因素分解
long long factor[100];//质因素分解结果(刚返回时时无序的)
int tol;//质因素的个数，编号0~tol-1
//找出一个因子
long long pollard_rho(long long x,long long c) {
    long long i = 1, k = 2;
    srand((long long) time(NULL));
    long long x0 = rand()%(x-1) + 1; long long y = x0;
    while(1) {
        i ++;
        x0 = (mul (x0,x0,x) + c)%x; long long d = __gcd(y - x0,x);
        if( d != 1 && d != x)return d;
        if(y == x0)return x;
        if(i == k){y = x0; k += k;}
    }
}
//对n进行素因子分解，存入factor. k设置为107左右即可
void findfac(long long n,int k){
    if(n == 1)return; if(Miller_Rabin(n)) {
    factor[tol++] = n;
    return; }
    long long p = n; int c = k;
    while( p >= n) p = pollard_rho(p,c--);//值变化，防止死循环k
    findfac(p,k);
    findfac(n/p,k);
}
//POJ 1811
//给出一个N(2 <= N < 2^54),如果是素数，输出"Prime",否则输出最小的素因子
int main() {
    int T;
    long long n;
    scanf("%d",&T);
    while(T--) {
        scanf("%lld",&n);
        if(Miller_Rabin(n))printf("Prime\n");
        else {
            tol = 0;
            findfac(n,107);
            long long ans = factor[0];
            for(int i = 1;i < tol;i++) ans = min(ans,factor[i]);
            printf("%lld\n",ans);
        }
    }
    return 0;
}
```

pell方程

- 形如$x^2 - dy^2 = 1$的不定方程是pell方程。如果d是完全平方数那么方程无解。
- 当找到一组最小特解以后其他的解可以利用矩阵迭代算出：$\begin{bmatrix} x_k \\ y_k \end{bmatrix} = \begin{bmatrix} x_1 & d \times y_1 \\ y_1 & x_1 \end{bmatrix}^{k-1} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$

c++在线程序（当n比较大的时候会爆longlong，小心），如果无解则没有输出，如果有解输出最小解。

```
typedef long long ll;
ll a[20000];
bool pell_minimum_solution(ll n,ll &x0,ll &y0){
    ll m=(ll)sqrt((double)n);
    double sq=sqrt(n);
    int i=0;
    if(m*m==n)return false;//当n是完全平方数则佩尔方程无解
    a[i++]=m;
    ll b=m,c=1;
    double tmp;
    do{
        c=(n-b*b)/c;
        tmp=(sq+b)/c;
        a[i++]=(ll)(floor(tmp));
        b=a[i-1]*c-b;
    }while(a[i-1]!=2*a[0]);
    ll p=1,q=0;
    for(int j=i-2;j>=0;j--){
        ll t=p;
        p=q+p*a[j];
        q=t;
    }
    if((i-1)%2==0){x0=p;y0=q;}
    else{x0=2*p*p+1;y0=2*p*q;}
    return true;
}

int main(){
    ll n,x,y;
    while(scanf("%lld",&n) == 1){
        if(pell_minimum_solution(n,x,y)){
            printf("%lld^2-%lld*%lld^2=1\t",x,n,y);
            printf("%lld-%lld=1\n",x*x,n*y*y);
        }
    }
    return 0;
}
```

java打表程序，得到是对应d下的最小特解。如果无解打表结果是"no solution"。注意是从1开始的。

```
import java.io.*;
import java.math.*;
import java.util.*;

public class Main {
    static long [] a = new long [1000];
    static BigInteger x;
    static BigInteger y;
    public static void main(String [] args) throws FileNotFoundException{
        FileReader fin = new FileReader ("data.in");
        File fout = new File("data.out");
        PrintStream pw = new PrintStream(fout);
        Scanner cin = new Scanner(fin);
        System.setOut(pw);
        while(cin.hasNext()){
            long n=cin.nextLong();
            if(pell_solution(n)){

                System.out.print("\""+x+"\",");
            }else{
                System.out.print("\"no solution\",");
            }
        }
    }
    public static boolean pell_solution(long D){
        double sq=Math.sqrt((double)D);
        long m=(long) Math.floor(sq);
        int i=0;
        if(m*m==D)return false;
        a[i++]=m;
        long b=m,c=1;
        double tmp;
        do{
            c=(D-b*b)/c;
            tmp=(sq+b)/c;
            a[i++]=(long)(Math.floor(tmp));
            b=a[i-1]*c-b;
        }while(a[i-1]!=2*a[0]);
        BigInteger p=new BigInteger("1");
        BigInteger q=new BigInteger("0");
        BigInteger t;
        for(int j=i-2;j>=0;j--){
            t=p;
            p=q.add(p.multiply(BigInteger.valueOf(a[j])));
            q=t;
        }
        if((i-1)%2==0){
            x=p;y=q;
        }else{
            x=BigInteger.valueOf(2).multiply(p).multiply(p).add(BigInteger.ONE);
            y=BigInteger.valueOf(2).multiply(p).multiply(q);
        }
        return true;
    }
}
```

线性递推

Berlekamp-Massey算法,用来计算x[]数列的递推式,根据递推式加快速矩阵幂求数列某一项

```
#include <bits/stdc++.h>
#define fir first
#define se second
#define pb push_back
#define ll long long
#define mp make_pair
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define all(x) (x).begin(),(x).end()
#define SZ(x) ((int)(x).size())
using namespace std;

typedef vector<ll> VI;
typedef pair<int,int> PII;
const int maxn=1e5+10;
const int maxm=1e6+10;
const int inf=0x3f3f3f3f;
const ll mod=1e9+7;
const double eps=1e-7;
ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0); for(;b;b>>=1)
{if(b&1)res=res*a%mod;a=a*a%mod;}return res;}
namespace linear_seq {
    const int N=10010;
    ll res[N],base[N],_c[N],_md[N];

    vector<int> Md;
    void mul(ll *a,ll *b,int k) {
        rep(i,0,k+k) _c[i]=0;
        rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
        for (int i=k+k-1;i>=k;i--) if (_c[i])
            rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[j]]-_c[i]*_md[Md[j]])%mod;
        rep(i,0,k) a[i]=_c[i];
    }
    ll solve(ll n,VI a,VI b) { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
//        printf("%d\n",SZ(b));
        ll ans=0,pnt=0;
        int k=SZ(a);
        assert(SZ(a)==SZ(b));
        rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
        Md.clear();
        rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
        rep(i,0,k) res[i]=base[i]=0;
        res[0]=1;
        while ((1ll<<pnt)<=n) pnt++;
        for (int p=pnt;p>=0;p--) {
            mul(res,res,k);
            if ((n>>p)&1) {
                for (int i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;
                rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%mod;
            }
        }
        rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
        if (ans<0) ans+=mod;
        return ans;
    }
    VI BM(VI s) {
        VI C(1,1),B(1,1);
        int L=0,m=1,b=1;
        rep(n,0,SZ(s)) {
            ll d=0;
            rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
            if (d==0) ++m;
            else if (2*L<=n) {
                VI T=C;
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                L=n+1-L; B=T; b=d; m=1;
            } else {
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                ++m;
            }
        }
        return C;
    }
    int gao(VI a,ll n) {
        VI c=BM(a);
        c.erase(c.begin());
        rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
        return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
    }
};

int n,k;
vector<ll> a,b;

int main(){
    while (~scanf("%d %d",&n,&k)){
        a.clear();
        b.clear();
        for (int i=0;i<n;i++){
            int num;
            scanf("%d",&num);
            b.pb(num);
        }
        for (int i=0;i<n;i++){
            int num;
            scanf("%d",&num);
            a.pb(num);
        }
        printf("%lld\n",linear_seq::solve(k-1,a,b));
    }
    return 0;
}
```

二次剩余

用于计算$x^2 \equiv n(mod\ p)$意义下的x的解,可用于计算数论意义下的平方根（负数也可以）

```
#define rep(i,l,r) for(int i=l;i<=r;i++)
#define per(i,r,l) for(int i=r;i>=l;i--)
#define ull unsigned long long
#define ll long long
#define N 100005
namespace random_gen{
    ull r=0x1234567890ABCDEF;
    ull f(){
        r=r*0x234567890FEDBCA1+0xABCFED0987654123;
        return r=r>>29|r<<35;
    }
}
namespace numbertheory{
    ll mulmod(ll a,ll b,ll p){
        ll s=0;
        while(b){
            if(b&1) s=(s+a)%p;
            b>>=1;a=(a+a)%p;
        }
        return s;
    }
    ll powmod(ll a,ll b,ll p){
        ll s=1;
        while(b){
            if(b&1) s=mulmod(s,a,p);
            b>>=1;a=mulmod(a,a,p);
        }
        return s;
    }
    ll Legendre(ll a,ll p){
        return powmod(a,p-1>>1,p);
    }
    namespace fieldextension{
        ll d,p;
        struct num{
            ll x,y;
        };
        num operator*(num a,num b){
            return (num){(mulmod(a.x,b.x,p)+mulmod(mulmod(a.y,b.y,p),d,p))%p,
(mulmod(a.x,b.y,p)+mulmod(a.y,b.x,p))%p};
        }
        num operator^(num a,ll b){
            num s=(num){1,0};
            while(b){
                if(b&1) s=s*a;
                b>>=1;a=a*a;
            }
            return s;
        }
    }
    ll quadratic_equation(ll n,ll p){
        if(!n) return 0;
        if(p==2) return 1;
        if(Legendre(n,p)==p-1) return -1;
        ll a,w;
        while(1){
            a=random_gen::f()%p;
            w=(mulmod(a,a,p)-n+p)%p;
```

```
                //printf("%lld %lld\n",a,w);
                if(Legendre(w,p)==p-1){
                    fieldextension::d=w;
                    fieldextension::p=p;
                    return ((fieldextension::num){a,1}^p+1>>1).x;
                }
            }
        }
    }
```

博弈

威佐夫博弈

`HDU 1527`:两堆石头，每次可以一堆中任取或者两堆取相同数量，取走最后一个石头获胜。

```
int main () {
    while (cin >> a >> b) {
        if (a > b) swap (a, b);
        long long k = b-a;
        if (a == (long long)(k*(sqrt(5)+1) / 2) ) cout << "0" << endl;
        else cout << "1" << endl;
    }
    return 0;
}
```

- 扩展 如果石头的堆数是奇数，规则同威佐夫博弈，那么可以按照NIM博弈做。

NIM博弈

sg函数

sg(state)==0表示先手必败，sg(state)!=0表示先手必胜。一个状态的sg值等于所有后继状态中第一个没有出现的值。

nim积

`HDU3404`：翻硬币游戏 每次选择一个矩形的四个角进行反转，保证右下角的一定要从开到关 问是否有必胜策略

```
#include<iostream>
#include<cstdio>
#include<cstring>
#define N 2000000
using namespace std;
int m[2][2]={0,0,0,1};
int Nim_Mult_Power(int x,int y){
    if(x<2)
        return m[x][y];
    int a=0;
    for(;;a++)
        if(x>=(1<<(1<<a))&&x<(1<<(1<<(a+1))))
```

```
            break;
    int m=1<<(1<<a);
    int p=x/m,s=y/m,t=y%m;
    int d1=Nim_Mult_Power(p,s);
    int d2=Nim_Mult_Power(p,t);
    return (m*(d1^d2))^Nim_Mult_Power(m/2,d1);
}
int Nim_Mult(int x,int y){
    if(x<y)
        return Nim_Mult(y,x);
    if(x<2)
        return m[x][y];
    int a=0;
    for(;;a++)
        if(x>=(1<<(1<<a))&&x<(1<<(1<<(a+1))))
            break;
    int m=1<<(1<<a);
    int p=x/m,q=x%m,s=y/m,t=y%m;
    int c1=Nim_Mult(p,s),c2=Nim_Mult(p,t)^Nim_Mult(q,s),c3=Nim_Mult(q,t);
    return (m*(c1^c2))^c3^Nim_Mult_Power(m/2,c1);
}
int main(){
    int t,n,x,y;
    scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        int ret=0;
        while(n--){
            scanf("%d%d",&x,&y);
            ret^=Nim_Mult(x,y);
        }
        if(ret)
            puts("Have a try, lxhgww.");
        else
            puts("Don't waste your time.");
    }
    return 0;
}
```

必败态扩充

`2014 西安H`：一个有向图，两人轮流移动棋子，在同一个点算B失败，死局算B赢，谁先无法移动算输。问先手是否必胜。 维护bob的必败态，从必败态不断bfs扩展。

```
#define maxn 105
#define maxm 5100005

int g[maxn][maxn];
int n, m;
int degree[maxn];//出度
int num[maxn][maxn];//是多少个必败态的前一状态
int dp[maxn][maxn][2];//alice bob 0:alice 1:bob

struct node {
    int x, y;
    int id;
};
queue <node> q;

bool bfs () {
    while (!q.empty ()) {
        node now = q.front (); q.pop ();
        if (now.id == 0) {//这一步是alice先手 上一步是bob先手
            for (int i = 1; i <= n; i++) if (g[i][now.y]) { //cout << i << endl;
                num[now.x][i]++;
                if (num[now.x][i] == degree[i]) {//alice在x bob走到的每一处都是必败态
                    dp[now.x][i][1] = 1;
                    q.push ((node) {now.x, i, 1});
                }
            }
        }
        else {//这一步是bob先手 上一步是alice先手
            for (int i = 1; i <= n; i++) if (g[i][now.x]) {
                if (dp[i][now.y][0])//已经选择过了 避免重复bfs
                    continue;
                dp[i][now.y][0] = 1;
                q.push ((node) {i, now.y, 0});
            }
        }
    }
    int v, u;//bob alice
    scanf ("%d%d", &v, &u);
    if (dp[u][v][1])
        return 0;
    return 1;
}

int main () {
    int t, kase = 0;
    scanf ("%d", &t);
    while (t--) {
        scanf ("%d%d", &n, &m);
        memset (num, 0, sizeof num);
        memset (g, 0, sizeof g);
        memset (degree, 0, sizeof degree);
        memset (dp, 0, sizeof dp);
        for (int i = 0; i < m; i++) {
            int u, v;
            scanf ("%d%d", &u, &v);
            g[u][v] = 1;
            degree[u]++;
        }
        while (!q.empty ())
            q.pop ();
        for (int i = 1; i <= n; i++) {//在同一个节点
            dp[i][i][0] = dp[i][i][1] = 1;
            q.push ((node) {i, i, 0});
            q.push ((node) {i, i, 1});
        }
        for (int i = 1; i <= n; i++) {//bob先手并且无路可走
            for (int j = 1; j <= n; j++) if (i != j) {
                if (degree[j] == 0) {
                    dp[i][j][1] = 1;
                    q.push ((node) {i, j, 1});
                }
            }
        }
        //bfs ();
        printf ("Case #%d: %s\n", ++kase, bfs () ? "Yes" : "No");
    }
    return 0;
}
```

树上删边博弈

`HDU 3094` : 树上的删边游戏,删掉一条边后只保留与根节点向连的部分,最后无法删边的人输. 一个有趣的结论:这个游戏中某个节点的sg值等于他所有孩子节点sg值+1的异或和,其中叶子节点的sg值等于0.

```
#define maxn 211111

struct node {
    int u, v, next;
}edge[maxn];
int head[maxn], cnt, n;
int sg[maxn];

void add_edge (int u, int v) {
    edge[cnt].u = u, edge[cnt].v = v, edge[cnt].next = head[u], head[u] = cnt++;
}

int dfs (int u, int father) {
    int ans = 0;
    for (int i = head[u]; i != -1; i = edge[i].next) {
        int v = edge[i].v;
        if (v == father)
            continue;
        ans ^= (dfs (v, u)+1);
    }
    return ans;
}
```

```
int main () {
    //freopen ("in.txt", "r", stdin);
    int t;
    scanf ("%d", &t);
    while (t--) {
        memset (head, -1, sizeof head);
        cnt = 0;
        scanf ("%d", &n);
        for (int i = 1; i < n; i++) {
            int u, v;
            scanf ("%d%d", &u, &v);
            add_edge (u, v);
            add_edge (v, u);
        }
        int ans = dfs (1, 0);
        if (ans)
            printf ("Alice\n");
        else
            printf ("Bob\n");
    }
    return 0;
}
```

矩阵

矩阵快速幂

```
struct matrix {
    long long a[maxn][maxn];
    matrix operator * (matrix gg) {
        matrix ans;
        memset (ans.a, 0, sizeof ans.a);
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                for (int l = 0; l < 2; l++) {
                    ans.a[i][j] += a[i][l]*gg.a[l][j];
                    ans.a[i][j] %= mod;
                }
            }
        }
        return ans;
    }
    void show () {
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++)
                cout << a[i][j] << " ";
            cout << endl;
        }
    }
};
```

```
matrix qpow (matrix a, long long kk) {
    matrix ans;
    int i, j;
    for(i = 0; i < 2; ++i)
        for(j = 0; j < 2; ++j)
            ans.a[i][j] = (i == j ? 1 : 0);
    for(; kk; kk >>= 1) {
        if (kk&1) ans = ans*a;
        a = a*a;
    }
    return ans;
}
```

线性筛(质数 欧拉函数 最小素因子 莫比乌斯函数)

```
bool vis[maxn];
int phi[maxn], prime[maxn], cnt, fac[maxn];
void init () {
    cnt = 0;
    memset (vis, 0, sizeof vis);
    mu[1] = 1;
    phi[1] = 1;
    for (int i = 2; i < maxn; i++) {
        if (!vis[i]) {
            prime[cnt++] = i;
            fac[i] = i;
            phi[i] = i-1;
            mu[i] = -1;
        }
        for (int j = 0; j < cnt; j++) {
            if (1LL*i*prime[j] >= maxn)
                break;
            vis[i*prime[j]] = 1;
            fac[i*prime[j]] = prime[j];
            if (i%prime[j] == 0) {
                phi[i*prime[j]] = phi[i] * prime[j];
                mu[i*prime[j]] = 0;
                break;
            }
            else {
                phi[i*prime[j]] = phi[i] * (prime[j]-1);
                mu[i*prime[j]] = -mu[i];
            }
        }
    }
}
```

欧拉函数

欧拉公式降幂

- 当$x > \varphi(m)$时，$a^x \% m = a^{\varphi(m)+x\%\varphi(m)} \% m$

求解欧拉函数

- 线性筛（参见线性筛）
- $O(\sqrt(n))$

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

积性函数求和

`HDU 6439`：首先，gcd(a{1},a{2}...,a_{n},m)=1的时候才能有解

我们记不是-1的a的gcd为g，并记录-1的数量num

如果g不为0，相当于问这num个变量有多少种取法，使得gcd为1

因为最后的gcd一定是g的因子，所以我们只需要对g的因子进行容斥

对于某个因子p,当前的答案就是1$^{\{num\}+2}${num}+...+(n/p)^{num}

考虑到num最多50，可以采用伯努利数的方法来求自然数幂和

如果g=0 就是问这num个变量有多少种取法，使得gcd为1

考虑到如果num=1，答案显然为phi(n)

所以可以尝试打num=2的表，可以总结出规律：

ans(n)=n$^{\{num\}*((p\_\{1\}}${num}-1)/p{1}^{num})*((p{2}$^{\{num\}-1)/p\_\{2\}}${num})....

这个函数与欧拉函数类似，显然，这也是个积性函数

之后就是用类似求欧拉函数前n项和的办法来就可以了

只需要改一改线性筛的部分以及将n*(n+1)/2改成1^{k+2}k+...+n^k即可

```
#include<bits/stdc++.h>
#define mp make_pair
#define fir first
#define se second
```

```
#define ll long long
#define pb push_back
#define LL long long
#pragma comment(linker, "/STACK:1024000000,1024000000")
using namespace std;
const int maxn=2e6+10;
const ll mod=1e9+7;
const ll MOD=mod;
const int maxm=1e6+10;
const double eps=1e-7;
const int inf=0x3f3f3f3f;
const double INF = 1e20;
const double pi = acos (-1.0);
const int N=100;
int gcd(int a,int b){
    return (b==0?a:gcd(b,a%b));
}
ll qpow(ll a,ll b,ll p){
    ll res=1;
    while (b){
        if (b&1) res=res*a%p;
        b>>=1;
        a=a*a%p;
    }
    return res;
}
//伯努利数（自然数幂和的预处理部分）
LL C[N][N];
LL B[N],Inv[N];
LL Tmp[N];
void Init()
{
    //预处理组合数
    for(int i=0; i<N; i++)
    {
        C[i][0] = C[i][i] = 1;
        if(i == 0) continue;
        for(int j=1; j<i; j++)
            C[i][j] = (C[i-1][j] % MOD + C[i-1][j-1] % MOD) % MOD;
    }
    //预处理逆元
    Inv[1] = 1;
    for(int i=2; i<N; i++)
        Inv[i] = (MOD - MOD / i) * Inv[MOD % i] % MOD;
    //预处理伯努利数
    B[0] = 1;
    for(int i=1; i<N; i++)
    {
        LL ans = 0;
        if(i == N - 1) break;
        for(int j=0; j<i; j++)
        {
            ans += C[i+1][j] * B[j];
            ans %= MOD;
        }
        ans *= -Inv[i+1];
        ans = (ans % MOD + MOD) % MOD;
        B[i] = ans;
    }
}

ll k,n;
ll a[maxn];

ll sum(ll n,ll k){
    //用伯努利数算1^k+2^k+..+n^k;
    if (k==0) return n;
    ll res=0,now=n+1;
    for (int i=1;i<=k+1;i++){
        res=(res+C[k+1][i]*B[k+1-i]%mod*now%mod)%mod;
        now=now*(n+1)%mod;
    }
    res=res*Inv[k+1]%mod;
    return res;
}
bool vis[maxn];
ll phi[maxn], prime[maxn], cnt, prime1[maxn];
ll Sum[maxn];
void pre (int Maxn,int k)
{
    cnt = 0;
    for (int i=0;i<=Maxn;i++)
        vis[i]=0;
    phi[1] = 1;
    for ( int i = 2; i <= Maxn; i++)
    {
        if (!vis[i])
        {
            prime[cnt++] = i;
            prime1[i]=qpow(i,k,mod);
            phi[i] = qpow(i,k,mod)-1;
        }
        for ( int j = 0; j < cnt; j++)
        {
            if ( 1LL*i*prime[j] > Maxn)
                break;
            vis[i*prime[j]] = 1;
            if (i%prime[j] == 0)
            {
                phi[i*prime[j]] = phi[i] * prime1[prime[j]]%mod;
                //cout<<i*prime[j]<<" "<<phi[i]<<" "<<prime[j]<<" "
<<phi[i*prime[j]]endl;
```

```
                break;
            }
            else
            {
                phi[i*prime[j]] = phi[i] * (prime1[prime[j]]-1)%mod;
            }
        }
    }
    Sum[0]=0;
    for (int i=1;i<=Maxn;i++)
        Sum[i]=(Sum[i-1]+phi[i])%mod;
}
map<long long ,long long> ma;
ll calc(long long n,int Maxn,int num){
    if(n<=Maxn) return Sum[n];
    if(ma.find(n)!=ma.end()) return ma[n];
    ll ans=sum(n,num);
    for(long long i=2,r;i<=n;i=r+1){
        r=n/(n/i);
        ans=(ans-1LL*calc(n/i,Maxn,num)*((r-i+1))%mod)%mod;
    }
   return ma[n]=(ans+mod)%mod;
}
int main(){
    int t;
    scanf("%d",&t);
    Init();
    int kase=0;
    while (t--){
        kase++;
        int num=0;
        scanf("%lld %lld",&k,&n);
        for (int i=1;i<=k;i++){
            scanf("%lld",&a[i]);
        }
        ll g=0;
        for (int i=1;i<=k;i++){
            if (a[i]==-1){
                num++;
            }
            else g=gcd(g,a[i]);
        }
        if (g!=0){
            //枚举g的因子 大力容斥
            vector<ll> v;
            ll temp=g;
            for (ll i=2;i*i<=g;i++){
                if (temp%i==0){
                    v.pb(i);
                    while(temp%i==0) temp/=i;
                }
            }
            if (temp>1) v.pb(temp);
            ll ans=0;
            for (int i=0;i<(1<<v.size());i++){
                ll d=1,flag=1;
                for (int  j=0;j<v.size();j++){
                    if (i&(1<<j)){
                        d*=v[j];
                        flag*=-1;
                    }
                }
                if (n>=d){
                    ans=(ans+flag*sum(n/d,num)%mod)%mod;
                }
            }
            ans=(ans+mod)%mod;
            printf("Case #%d: %lld\n",kase,ans);
        }
        else{
            int Maxn=pow(n,2.0/3.0);
            pre(Maxn,num);
            ma.clear();
            ll ans=calc(n,Maxn,num);
            ans=(ans+mod)%mod;
            printf("Case #%d: %lld\n",kase,ans);
        }
    }
    return 0;
}
```

组合数学(容斥&&反演等)

一些结论

- 错排公式：f(i)表示i个元素错排个数，那么有$f(i) = (i-1)*(f(i-1)+f(j-2)) = if(i-1)+(-1)^i$.
- $\sum_{i=0}^{n} i * C_n^i = n * 2^{n-1}$.
- $\sum_{i=0}^{n} (C_n^i)^2 = C_{2n}^n$.
- C(n,m) 的奇偶性判断： **if ((n & m) == m) 为奇数，否则为偶**

枚举子集

```
for (int x = S; x; x = (x-1)&S) {...}
```

第二类斯特林数

- S(n,m)表示n个不同的数拆成m个非空的集合的方案数.
- 重要性质：$m^n = \sum_{k=0}^{min(n,m)} A_n^k * S(n,k)$.

`codeforces round #463 E`:求$\sum_{i=1}^{n} C_n^i * i^k, k \le 5000, n \le 1e9$.

$$n^{\underline{j}} = A_n^j$$

$$\left\{k, j\right\} = S(k, j)$$

$$\begin{aligned}
&\sum_{i=0}^{n}\binom{n}{i} \cdot i^k \\
&= \sum_{i=0}^{n}\binom{n}{i} \cdot \sum_{j=0}^{k}\left\{{k \atop j}\right\} i^{\underline{j}} \\
&= \sum_{i=0}^{n}\binom{n}{i} \cdot \sum_{j=0}^{k}\left\{{k \atop j}\right\} \cdot j!\binom{i}{j} \\
&= \sum_{i=0}^{n} \frac{n!}{(n-i)!} \cdot \sum_{j=0}^{k}\left\{{k \atop j}\right\} \cdot \frac{1}{(i-j)!} \\
&= \sum_{i=0}^{n} \sum_{j=0}^{k} \frac{n!}{(n-i)!} \cdot \left\{{k \atop j}\right\} \cdot \frac{1}{(i-j)!} \\
&= n! \sum_{i=0}^{n} \sum_{j=0}^{k} \left\{{k \atop j}\right\} \cdot \frac{1}{(n-i)!} \cdot \frac{1}{(i-j)!} \\
&= n! \sum_{i=0}^{n} \sum_{j=0}^{k} \left\{{k \atop j}\right\} \cdot \binom{n-j}{n-i} \cdot \frac{1}{(n-j)!} \\
&= n! \sum_{j=0}^{k} \left\{{k \atop j}\right\} \cdot \frac{1}{(n-j)!} \sum_{i=0}^{n} \cdot \binom{n-j}{n-i} \\
&= \sum_{j=0}^{k} \left\{{k \atop j}\right\} \cdot n^{\underline{j}} \cdot 2^{n-j}
\end{aligned}$$

组合计数

`2016四川省赛H`：给出一个树，每一个树边都有$2 * c_i$条，求从1出发回到1的欧拉回路个数。

- 按照每一条边以及边之间的相对次序计算对答案的贡献。对于父亲节点u，他到某一个儿子节点的边数如果是c_i，那么$ans * (2c_i)!$。然后所有的这些儿子节点的路径经历的相对次序又对答案产生一部分贡献，所以假设某个节点有now对出去的路径（包括通向他父亲的路径对数-1），那么就要$ans * now!$，表示这些路径出现的相对次序的排列。但是注意到u->v的所有的边的对数已经在$(2c_i)!$中计算了这$2c_i$对边的相对次序，所以最后的答案需要除以所有的$c_i!$。

```
long long fac[maxn*10], rev[maxn*10];

long long qpow (long long a, long long b) {
    if (b == 0) return 1;
    long long ans = qpow (a, b>>1);
    ans = ans*ans%mod;
    if (b&1) ans = ans*a%mod;
    return ans;
}

void init () {
    int Max = 2000005;
    fac[0] = 1;
    for (int i = 1; i < Max; i++) {
        fac[i] = fac[i-1]*i%mod;
    }
    rev[Max-1] = qpow (fac[Max-1], mod-2);
    for (int i = Max-2; i >= 0; i--) {
        rev[i] = rev[i+1]*(i+1)%mod;
    }
}

int n;
int a[maxn];
struct node {
    int v, w, next;
}edge[maxn<<1];
int head[maxn], cnt;

void add_edge (int u, int v, int w) {
    edge[cnt].v = v, edge[cnt].w = w, edge[cnt].next = head[u], head[u] = cnt++;
}

long long ans;
bool vis[maxn];

void bfs () {
    queue <int> gg;
    while (!gg.empty ()) gg.pop ();
    gg.push (1);
    memset (vis, 0, sizeof vis);
    vis[1] = 1;
    while (!gg.empty ()) {
        int u = gg.front (); gg.pop ();
        for (int i = head[u]; i != -1; i = edge[i].next) {
            int v = edge[i].v, w = edge[i].w;
            if (vis[v] == 1) continue;
            ans *= fac[2*w]; ans %= mod;
            if (w >= 2) {
                ans *= rev[w-1];
                ans %= mod;
                a[v] += w-1;
            }
            ans *= rev[w];
            ans %= mod;
            a[u] += w;
            gg.push (v);
            vis[v] = 1;
        }
        ans = ans*fac[a[u]]%mod;
    }
    cout << ans << "\n";
}

int main () {
    init ();
    scanf ("%d", &n);
    memset (a, 0, sizeof a);
    cnt = 0;
    memset (head, -1, sizeof head);
    for (int i = 0; i < n-1; i++) {
        int u, v, w;
        scanf ("%d%d%d", &u, &v, &w);
        add_edge (u, v, w);
        add_edge (v, u, w);
    }
    ans = 1;
    bfs ();
    return 0;
}
```

容斥原理

`HDU 1695` : 求[1,b],[1,d]中gcd==k的pair。 相当于求[1,b/k],[1,d/k]中互质的pair。

```
int c, d, a, b, k;
vector <int> gg[maxn];
bool is_prime[maxn];
int cnt, prime[maxn];

void get_prime () { //分解1-100000所有的数
    for (int i = 1; i <= 100000; i++)
        gg[i].clear();
    memset (is_prime, 1, sizeof is_prime);
    is_prime[0] = is_prime[1] = 0;
    cnt = 0;
    for (int i = 2; i <= 100000; i++) {
        if (is_prime[i]) {
            prime[++cnt] = i;
            gg[i].push_back (i);
            for (int j = i+i; j <= 100000; j += i) {
                is_prime[j] = 0;
                gg[j].push_back (i);
            }
        }
    }
}

int count (int num, long long &mul, int x) {
    mul = 1;
    int ans = 0;
    int Max = gg[x].size ();
    for (int i = 1; i <= Max; i++, num >>= 1) {
        if (num&1) {
            mul *= gg[x][i-1];
            ans++;
        }
    }
    return ans;
}
```

```
long long solve () {//[1,b], [1,d]
    long long ans = 0;
    for (int x = 1; x <= d; x++) { //求[1,min(b,x)]中和x不互质的个数
        int r = min (b, x);
        long long cur = 0, mul;
        for (int i = 1; i < (1<<gg[x].size ()); i++) {
            int num_of_1 = count (i, mul, x); //二进制i中1的个数
            if (num_of_1&1)
                cur += r/mul;
            else cur -= r/mul;
        }
        ans += (r-cur);
    }
    return ans;
}

int main () {
    int t, kase = 0;
    cin >> t;
    get_prime ();
    while (t--) {
        cin >> a >> b >> c >> d >> k;
        if (k == 0) {
            printf ("Case %d: 0\n", ++kase);
            continue;
        }
        b /= k, d /= k;
        if (d < b)
            swap (d, b);
        printf ("Case %d: %lld\n", ++kase, solve ());
    }
    return 0;
}
```

`NEU 1660`: $f(n)=f(n-1)+f(n-2)(3\leq n),f(1)=f(2)=1$,计算$\sum_{i=1\&\&gcd(i,n)==1}^{n}f(i)$. 题目可以转化为求前f(1)+f(2)+..+f(n)-Σ f(i) (gcd (i, n) != 1) . 对于利用容斥原理，就可以发现只需要搞定f(k)+f(2k)+f(3k)+..+f(tk) (t = n/k) . 对于前面部分的f(1)+f(2)+f(3)+...+f(n)也是一样 . 这么一个下标等差的斐波那契数列的和可以二分求也可以纯矩阵求.

```
#define pb push_back
using namespace std;
const long long mod = 1e9+7;

struct m {
    long long a[3][3];
    m operator + (m b) const {
        m ans;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                ans.a[i][j] = a[i][j]+b.a[i][j];
                ans.a[i][j] %= mod;
            }
        }
        return ans;
    }
    m operator * (m b) const {
        m ans;
        memset (ans.a, 0, sizeof ans.a);
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                for (int k = 0; k < 3; k++) {
                    ans.a[i][j] += a[i][k]*b.a[k][j]%mod;
                    ans.a[i][j] %= mod;
                }
            }
        }
        return ans;
    }
    void show () {
        cout << ".........." << endl;
        for (int i = 0; i < 3; i++) {
            for (int  j= 0; j < 3; j++)
                cout << a[i][j] << " "; cout << endl;
        }
        cout << ".........." << endl;
    }
};
long long n;

m qpow (m a, long long b) {
    if (b == 1)
        return a;
    m ans = qpow (a, b>>1);
    ans = ans*ans;
    if (b&1)
        ans = ans*a;
    return ans;
}

long long cal (long long k) { //计算f(k)+f(2k)+f(3k)+..+f(tk),tk<=n
    long long xx[3][3] = {{0,1,0}, {1,1,0}, {0,0,1}}, yy[3][3] = {{1,0,1}, {0,1,0},
{0,0,1}};
    m x;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++)
            x.a[i][j] = xx[i][j];
    }
    m y;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++)
```

```
            y.a[i][j] = yy[i][j];
    }
    long long tot = n/k;
    m cur = qpow (x, k); cur = cur*y;
    m ans = qpow (cur, tot);
    return ans.a[1][2]%mod;
}

vector <long long> fac;
void get_fac () { //分解n
    fac.clear ();
    long long num = n;
    for (long long i = 2; i*i <= num; i++) {
        if (num%i == 0) {
            fac.pb (i);
            while (num%i == 0)
                num /= i;
        }
    }
    if (num > 1)
        fac.pb (num);
    return ;
}

int num_of_1 (long long num, long long &mul, long long Max) { //统计 1 的个数
    mul = 1;
    int ans = 0;
    for (long long bit = 1, i = 0; bit <= Max; bit <<= 1, i++) {
        if (num&bit) {
            ans++;
            mul *= fac[i];
        }
    }
    return ans;
}

void work () {
    long long ans = cal (1);
    long long bit = fac.size ();
    long long Max = (1<<bit)-1;
    for (long long i = 1; i <= Max; i++) {
        long long mul = 0; //这个数字对应的素因子乘积
        int cnt = num_of_1 (i, mul, Max); //需要加上还是减去
        if (cnt&1) { //加上
            ans -= cal (mul);
            while (ans < 0)
                ans += mod;
            ans %= mod;
        }
        else { //减去
            ans += cal (mul);
```

```
                ans %= mod;
            }
        }
        printf ("%lld\n", ans);
}

int main () {
    while (scanf ("%lld", &n) == 1) {
        get_fac ();
        work ();
    }
    return 0;
}
```

`UVALive 7040` : n朵花,从m种颜色中选择k种颜色使得相邻花不同色的方案数. 考虑k种颜色涂色时相邻不同色的方案有$k*(k-1)^(n-1)$,这之中含有实际只用了1种颜色,2种颜色....k-1种颜色,需要把他们减去.所以减去1种颜色没有用的情况,但是这里面包含了2种颜色没有用的情况,所以要加上2中颜色没有用的情况....这么迭代下去就是容斥,把多的减掉少的加上.

```
long long f[maxn];//逆元
void init(){
    f[1] = 1;
    for (int i = 2; i <= 1000000; i++){
        f[i] = (long long)(mod - mod/i) * f[mod % i] % mod;
    }
}

long long n, k, m;
long long c[maxn];

long long qpow (long long a, long long b) {
    a %= mod;
    if (b == 0)
        return 1;
    long long ans = qpow (a, b>>1);
    ans = ans*ans%mod;
    if (b&1)
        ans = ans*a%mod;
    return ans;
}

int main () {
    init ();
    int t, kase = 0;
    scanf ("%d", &t);
    while (t--) {
        scanf ("%lld%lld%lld", &n, &m, &k);
        printf ("Case #%d: ", ++kase);
        if (k == 1 && n == 1) {
            printf ("%lld\n", m);
```

```
                continue;
            }
            if (k == 1 && n > 1) {
                printf ("0\n");
                continue;
            }
            long long ans = m%mod;
            for (long long i = 2; i <= k; i++) {
                ans = ans*(m-i+1)%mod*f[i]%mod;
            }
            c[1] = k%mod;
            for (long long i = 2; i <= k; i++) {
                c[i] = c[i-1]*(k-i+1)%mod*f[i]%mod;
            }
            long long res = k*qpow (k-1, n-1) % mod;
            for (long long i = 1; i < k; i++) {
                long long id = ((i&1) ? -1 : 1);
                long long cur = c[i]*(k-i)%mod*qpow (k-i-1, n-1)%mod;
                cur *= id;
                if (cur < 0)
                    cur += mod;
                res += cur;
                res %= mod;
            }
            printf ("%lld\n", ans*res%mod);
        }
        return 0;
    }
```

莫比乌斯反演

两种表述形式：

1. 若$F(n)=\sum_{d|n}f(d)$，那么$f(n)=\sum_{d|n}\mu(d)F(n/d)$;
2. 若$F(n)=\sum_{n|d}f(d)$，那么$f(n)=\sum_{n|d}\mu(d/n)F(d)$.

莫比乌斯函数的一些性质：

1. 如果n=1，$\sum_{d|n}\mu(d)=1$;如果n>1,$\sum_{d|n}\mu(d)=0$;
2. $\sum_{d|n}\frac{\mu(d)}{d}=\frac{\varphi(n)}{n}$
3. $n=\sum_{d|n}\varphi(d)$

`HDU 1695`：求[1,b],[1,d]中gcd==k的pair。 假设f(x)表示gcd为x的对数，F(x)表示gcd为x倍数的对数，那么显然f(x)和F(x)满足莫比乌斯反演的第二种描述。

```
long long a, b, c, d, k;
long long prime[maxn], cnt;
bool vis[maxn];
int mu[maxn];

void get_mu () {
    cnt = 0;
    memset (vis, 0, sizeof vis);
    mu[1] = 1;
    for (long long i = 2; i <= 100000; i++) {
        if (!vis[i]) {
            prime[cnt++] = i;
            mu[i] = -1;
        }
        for (long long j = 0; j < cnt; j++) {
            if (i*prime[j] > 100000)
                break;
            vis[i*prime[j]] = 1;
            if (i%prime[j] == 0) {
                mu[i*prime[j]] = 0;
                break;
            }
            else {
                mu[i*prime[j]] = -mu[i];
            }
        }
    }
}

long long solve (long long n, long long m) {
    long long ans = 0;
    for (long long i = 1; i <= n; i++) {
        ans += mu[i]*((long long)(n/i) * (long long)(m/i));
    }
    return ans;
}

int main () {
    //freopen ("in.txt", "r", stdin);
    get_mu ();
    int t, kase = 0;
    scanf ("%d", &t);
    while (t--) {
        scanf ("%lld%lld%lld%lld%lld", &a, &b, &c, &d, &k);
        if (k == 0) {
            printf ("Case %d: 0\n", ++kase);
            continue;
        }
        b /= k; d /= k;
        if (b > d)
            swap (b, d);
        printf ("Case %d: %lld\n", ++kase, solve (b, d)-solve (b, b)/2);
    }
    return 0;
}
```

分块加速

- 一个常见的优化：在计算$\sum_{i=1}^{n}\frac{n}{i}$时，可以用如下的分段加速：

```
int j = 1;
for (int i = 1; i <= n; i = j+1) {
    j = n/(n/i);
}
```

其中的i到j结果是相同的.

Lucas定理

- 大组合数对p（p是质数）取模,复杂度$O(plgp)$,注意预处理%p的阶乘数组fac。

```
long long Lucas (long long n, long long m, long long p) {
    long long ret=1;
    while (n&&m) {
        long long a=n%p,b=m%p;
        if (a < b) return 0;
        ret = (ret*fac[a]* qpow (fac[b]*fac[a-b]%p,p-2,p))%p;
        n /= p;
        m /= p;
    }
    return ret;
}
```

- 扩展lucas定理，此时p不必为质数。

```
long long Inv (long long a,long long n) {
    long long d,x,y;
    d = ex_gcd (a, n, x, y);
    return d==1?(x+n)%n:-1;
}
long long Fac (long long n,long long p,long long pr){
    if(n==0) return 1;
    long long re=1;
    for(long long i=2;i<=pr;i++) if(i%p) re=re*i%pr;
    re=qpow(re,n/pr,pr);
    long long r=n%pr;
    for(long long i=2;i<=r;i++) if(i%p) re=re*i%pr;
    return re*Fac(n/p,p,pr)%pr;

}
long long C(long long n,long long m,long long p,long long pr){
    if(n<m) return 0;
    long long x=Fac(n,p,pr),y=Fac(m,p,pr),z=Fac(n-m,p,pr);
    long long c=0;
    for(int i=n;i;i/=p) c+=i/p;
    for(int i=m;i;i/=p) c-=i/p;
    for(int i=n-m;i;i/=p) c-=i/p;
    long long a=x*Inv(y,pr)%pr*Inv(z,pr)%pr*qpow(p,c,pr)%pr;
    return a*(MOD/pr)%MOD*Inv(MOD/pr,pr)%MOD;
}
long long lucas (long long n,long long m){
    long long x=MOD,re=0;
    for(long long i=2;i<=x;i++) if(x%i==0){
        long long pr=1;
        while(x%i==0) x/=i,pr*=i;
        re=(re+C(n,m,i,pr))%MOD;
    }
    return re;
}
```

二分&&迭代&&三分

高斯消元

- xor方程

```
int a[maxn][maxn];
int x[maxn*maxn]; //解集
int equ, var;

int Gauss () {
    //equ个方程 var个未知数
    //-1:无解 0:唯一解  其他:返回自由元个数
    int max_r, col, k; //最大的数在的行  当前处理的列
    int cnt = 0; //自由元个数
    for (k = 0, col = 0; k < equ && col < var; k++, col++) {
        max_r = k;
        for (int i = k+1; i < equ; i++) { //找到最大的数所在的行
            if (abs (a[i][col]) > abs (a[max_r][col]))
                max_r = i;
        }
        if (max_r != k) { //交换最大的数所在的行和当前行
            for (int i = 0; i <= var; i++)
                swap (a[k][i], a[max_r][i]);
        }
        if (a[k][col] == 0) { //判断自由元  处理当前行的下一列
            k--;
            //free[cnt++] = col;
            continue;
        }
        for (int i = k+1; i < equ; i++) {
            if (a[i][col]) {
                for (int j = col; j <= var; j++) {
                    a[i][j] ^= a[k][j];
                }
            }
        }
    }

    for (int i = k; i < equ; i++) { //判断无解情况
        if (a[i][col])
            return -1;
    }
    if (k < var)
        return var - k; //返回自由元个数

    for (int i = var-1; i >= 0; i--) { //回代
        x[i] = a[i][var];
        for (int j = i+1; j < var; j++)
            x[i] ^= (a[i][j] && x[j]);
    }
    return 0;
}
```

- 浮点数方程组
- 整数方程组

线性基

概念：给出R维向量的一组点，求得的线性基进行线性组合可以表示任何一个点。

XOR线性基

求一组xor(模2)意义下的线性基

```
void xor_liner_basis (long long *aa, int n, long long *b) {
//求解一组xor线性基  一些线性基线性组合可以表示a中任意一个元素
//初始数组(0...n-1) 数组大小  返回的线性基
//b[i]表示最高位1是i位的线性基元素  如果这一位没有线性基元素就是-1
//b[63]表示能不能表示0  是的话为1  否则为-1
    vector <bool> vis (n+5, 0);
    vector <long long> a (n+5); b[63] = -1;
    for (int i = 0; i < n; i++) a[i] = aa[i];
    for (int i = 62; i >= 0; i--) {
        long long bit = (1LL<<i); int pos = -1;
        for (int j = 0; j < n; j++) if (!vis[j] && (a[j]&bit)) {
            pos = j; break;
        }
        if (pos == -1) {b[i] = -1; continue;}
        b[i] = pos; vis[pos] = 1;
        for (int j = 0; j < n; j++) if (j != pos && (a[j]&bit)) {
            a[j] ^= a[pos]; if (!a[j]) b[63] = 1;
        }
    }
    for (int i = 0; i <= 62; i++) if (b[i] != -1) b[i] = a[b[i]];
}
```

一些应用：

- 从n个数字中选择一些和x异或起来最大。

```
long long solve (long long *a, int n, long long *b, long long x) {
//数组 数组的元素 辅助数组 需要用来异或的值 返回异或之后的最大值
    xor_liner_basis(a, id, b);
    long long ans = x;
    for (int i = 62; i >= 0; i--) {
        long long bit = (1LL<<i);
        if (!(ans&bit)) { //这一位是0 需要选择线性基
            if (b[i] == -1) continue;
            else {
                ans ^= b[i];
            }
        }
    }
    return ans;
}
```

- `BZOJ 2460` n个数字对应一个价值，求一种价值最大的取法使得不存在xor为0的数字子集：求出线性基之后按照每一位取一个价值最大的1；

```
//pair<long long,long long> a[maxn]用来存放所有的数字和对应的价值
long long solve () {
    long long ans = 0; memset (vis, 0, (n+5)*sizeof vis[0]);
    for (int i = 62; i >= 0; i--) {
        long long Max = -INF, pos;
        for (int j = 0; j < n; j++) if ((a[j].fi&(1LL<<i)) && a[j].se > Max &&
!vis[j]) {
            Max = a[j].se;
            pos = j;
        }
        if (Max != -INF) {
            vis[pos] = 1;
            ans += Max;
            long long tmp = a[pos].fi;
            for (int j = 0; j < n; j++) if ((a[j].fi&(1LL<<i)) && !vis[j]) {
                a[j].fi ^= tmp;
            }
        }
    }
    return ans;
}
```

- `HDU 3949` n个数字，求xor第k小：求出线性基之后按位处理；

```
long long solve () {
    //xor第k小 无解返回-1
    long long k; scanf ("%lld", &k);
    if (b[63] == 1) {
        if (k == 1) return 0; //如果线性基能表示0 那么最小值是0
        k--;
    }
    long long res = 0, cnt = 0;
    for (int i = 0; i < 63; i++) if (b[i] != -1) cnt++;
    if (k >= (1LL<<cnt)) return -1; //无解
    for (int i = 62; i >= 0; i--) if (b[i] != -1) {
        cnt--;
        if (k >= (1LL<<cnt)) res ^= b[i], k -= (1LL<<cnt);
    }
    return res;
}
```

- n个点m条边的图，求xxx的路径，**特征是路径的权值等于经过点（边）的xor**：因为xor的性质，一般把图拆成一个生成树和若干个环，再用线性基进行处理； `BZOJ 2115` 最大xor从1到n路径。预处理生成树1到所有点的xor值，把所有基环的xor值找出来，相当于求生成树1到n的xor值和若干个基环xor以后的最大值。 `2015南阳CCPC E` 最大xor环。预处理所有基环的xor，相当于求一些值的最大xor结果。 `codeforces 724G` 所有的$(u, v, w) * w$从u到v路径xor是w的三元组和w的乘积。按位计算贡献。

卡特兰数

`母问题` 长度为2n的合法括号序列有多少种.

`变形问题`:

1. 从(1,1)走到(n,n),只走右下三角形的方案数;
2. 出栈顺序计数;
3. 排队问题:现在有2n个人，他们身高互不相同，他们要成两排，每一排有n个人，并且满足每一排必须是从矮到高，且后一排的人要比前一排对应的人要高，问有多少种排法;
4. 二叉树计数:n个节点的二叉树个数.

- 卡特兰数列第n项为$C(n) = C_{2n}^{n} - C_{2n}^{n-1} = \frac{1}{n+1} C_{2n}^{n}$. (1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190...)
- 求n对括号的所有合法序列中第i位是左/右括号的括号序列数。

枚举这个左/右括号和与之匹配的另一个，那么内部外部是两个子卡特兰序列，计算即可。

模方程相关

多元线性同余方程

定义：形如a1x1+a2x2+...+anxn+b≡0（mod m）的同余方程称为多元线性同余方程。

定理：多元线性同余方程a1x1+a2x2+...+anxn+b≡0（mod m）有解的充分必要条件是：

gcd（a1,a2,...,an,m）| b.若方程有解，则解(模m的剩余类)的个数为m^(n-1) ×gcd（a1,a2...,an,m）.

中国剩余定理

- 对于方程组$x \equiv a_i (\% m_i)$，其中m_i都是不同的质数，求出$x \% M (M = m_1 * \cdots * m_p)$

注意使用快速乘法，可能会爆longlong

```
long long mul (long long a, long long b, long long mod) {
    if (b == 0)
        return 0;
    long long ans = mul (a, b>>1, mod);
    ans = ans*2%mod;
    if (b&1) ans += a, ans %= mod;
    return ans;
}

long long china (int n, long long *a, long long *m) {
    long long M = 1, d, y, x = 0;
    for (int i = 0; i < n; i++) M *= m[i];
    for (int i = 0; i< n; i++) {
        long long w = M/m[i];
        gcd (m[i], w, d, d, y);
        x = (x+ mul (mul (y, w, M), a[i], M)) % M;
    }
    return (x+M) % M;
}
```

- 扩展中国剩余定理，模数不必互质

```
//参数表示除数数组 模数数组 同余方程数量
//无解返回-1 注意是否使用快速乘法
long long solve (long long *a, long long *r, int n) {
    long long M = a[1], R = r[1], x, y, d;
    for (int i = 2; i <= n; i++) {
        d = ex_gcd (M,a[i],x,y);
        if ((R-r[i])%d != 0) return -1;  //无解
        x = (R-r[i])/d*x%a[i]; //这才是真正的x，记得模a[i]。
        R -= x*M;   //特解x0，新的余数。
        M = M/d*a[i]; //新的模数。
        R %= M;
    }
    return (R%M+M)%M;
}
```

FFT&&(NTT)

原根

- 原根一般不会很大，以下代码可以求10^9以内的数的原根

```
int P;
const int NUM = 32170;
int prime[NUM/4];
bool f[NUM];
int pNum = 0;
void getPrime () {//线性筛选素数
```

```
    for (int i = 2; i < NUM; ++ i) {
        if (!f[i]) {
            f[i] = 1;
            prime[pNum++] = i;
        }
        for (int j = 0; j < pNum && i*prime[j] < NUM; ++ j) {
            f[i*prime[j]] = 1;
            if (i%prime[j] == 0) {
                break;
            }
        }
    }
}
long long getProduct(int a,int b,int P) {//快速求次幂mod
    long long ans = 1;
    long long tmp = a;
    while (b)
    {
        if (b&1)
        {
            ans = ans*tmp%P;
        }
        tmp = tmp*tmp%P;
        b>>=1;
    }
    return ans;
}

bool judge (int num) {//求num的所有的质因子
    int elem[1000];
    int elemNum = 0;
    int k = P - 1;
    for (int i = 0; i < pNum; ++ i) {
        bool flag = false;
        while (!(k%prime[i])) {
            flag = true;
            k /= prime[i];
        }
        if (flag) {
            elem[elemNum ++] = prime[i];
        }
        if (k == 1) {
            break;
        }
        if (k/prime[i]<prime[i]) {
            elem[elemNum ++] = prime[i];
            break;
        }
    }
    bool flag = true;
    for (int i = 0; i < elemNum; ++ i) {
        if (getProduct (num, (P-1)/elem[i], P) == 1) {
            flag = false;
            break;
        }
    }
    return flag;
}

int main()
{

    getPrime();
    while (cin >> P)
    {
        for (int i = 2;;++i)
        {
            if (judge(i))
            {
                cout << i<< endl;
                break;
            }
        }
    }
    return 0;
}
```

- 常用的原根表格(g就是原根)

$r * 2^k + 1$	r	k	g
3	1	1	2
5	1	2	2
17	1	4	3
97	3	5	5
193	3	6	5
257	1	8	3
7681	15	9	17
12289	3	12	11
40961	5	13	3
65537	1	16	3
786433	3	18	10
5767169	11	19	3
7340033	7	20	3
23068673	11	21	3
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
998244353			3
1004535809	479	21	3
2013265921	15	27	31
2281701377	17	27	3
3221225473	3	30	5
75161927681	35	31	3
77309411329	9	33	7
206158430209	3	36	22
2061584302081	15	37	7
2748779069441	5	39	3
6597069766657	3	41	5

$r*2^k+1$	r	k	g
39582418599937	9	42	5
79164837199873	9	43	5
263882790666241	15	44	7
1231453023109121	35	45	3
1337006139375617	19	46	3
3799912185593857	27	47	5
4222124650659841	15	48	19
7881299347898369	7	50	6
31525197391593473	7	52	3
180143985094819841	5	55	6
1945555039024054273	27	56	5
4179340454199820289	29	57	3

####关于生成函数的FFT

- 用于计算多项式$A(x)$和$B(x)$相乘后每一个次数对应的系数,其中:$A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$; $B(x) = b_0 + b_1x + b_2x^2 + \cdots + b_nx^n$
- 下标表示次数, 复数的实部表示系数.可能出现精度误差, 有时候需要换成long double.

`HDU 1402`:高精度整数相乘

```
#include <bits/stdc++.h>
using namespace std;
#define pi acos (-1)
#define maxn 200010

struct plex {
    double x, y;
    plex (double _x = 0.0, double _y = 0.0) : x (_x), y (_y) {}
    plex operator + (const plex &a) const {
        return plex (x+a.x, y+a.y);
    }
    plex operator - (const plex &a) const {
        return plex (x-a.x, y-a.y);
    }
    plex operator * (const plex &a) const {
        return plex (x*a.x-y*a.y, x*a.y+y*a.x);
    }
};
```

```
void change (plex *y, int len) {
    int i, j, k;
    for(i = 1, j = len / 2; i < len - 1; i++) {
        if (i < j) swap(y[i], y[j]);
        k = len / 2;
        while (j >= k) {
            j -= k;
            k /= 2;
        }
        if (j < k) j += k;
    }
}

void fft(plex y[],int len,int on)
{
    change(y,len);
    for(int h = 2; h <= len; h <<= 1)
    {
        plex wn(cos(-on*2*pi/h),sin(-on*2*pi/h));
        for(int j = 0;j < len;j+=h)
        {
            plex w(1,0);
            for(int k = j;k < j+h/2;k++)
            {
                plex u = y[k];
                plex t = w*y[k+h/2];
                y[k] = u+t;
                y[k+h/2] = u-t;
                w = w*wn;
            }
        }
    }
    if(on == -1)
        for(int i = 0;i < len;i++)
            y[i].x /= len;
}
char a[maxn], b[maxn];
plex x1[maxn], x2[maxn];
int ans[maxn];

int main () {
    while (scanf ("%s%s", a, b) == 2) {
        int len = 2, l1 = strlen (a), l2 = strlen (b);
        while (len < l1*2 || len < l2*2)
            len <<= 1;
        for (int i = 0; i < l1; i++) {
            x1[i] = plex (a[l1-1-i]-'0', 0);
        }
        for (int i = l1; i < len; i++)
            x1[i] = plex (0, 0);
        for (int i = 0; i < l2; i++) {
            x2[i] = plex (b[l2-1-i]-'0', 0);
        }
        for (int i = l2; i < len; i++)
            x2[i] = plex (0, 0);
        fft (x1, len, 1);
        fft (x2, len, 1);
        for (int i = 0; i < len; i++)
            x1[i] = x1[i]*x2[i];
        fft (x1, len, -1);
        for (int i = 0; i < len; i++) {
            ans[i] = (int)(x1[i].x+0.5);
        }
        for (int i = 0; i < len; i++) {
            if (ans[i] >= 10) {
                ans[i+1] += ans[i]/10;
                ans[i] %= 10;
            }
        }
        len = l1+l2-1;
        while (ans[len] <= 0 && len > 0)
            len--;
        for (int i = len; i >= 0; i--) {
            printf ("%d", ans[i]);
        }
        printf ("\n");
    }
    return 0;
}
```

关于卷积的FFT

`HDU` 计算$F(x) = \sum_{i=1}^{n-1} F(i) * a(n-i)\%$(a数组已知)

- cdq分治计算所有的F值。

```
using namespace std;
#define mod 313
#define pi acos(-1.0)
#define maxn 600005

struct plex {
    double x, y;
    plex (double _x = 0.0, double _y = 0.0) : x (_x), y (_y) {}
    plex operator + (const plex &a) const {
        return plex (x+a.x, y+a.y);
    }
    plex operator - (const plex &a) const {
        return plex (x-a.x, y-a.y);
    }
    plex operator * (const plex &a) const {
        return plex (x*a.x-y*a.y, x*a.y+y*a.x);
    }
}x1[maxn], x2[maxn];

void change (plex *y, int len) {
    int i, j, k;
    for(i = 1, j = len / 2; i < len - 1; i++) {
        if (i < j) swap(y[i], y[j]);
        k = len / 2;
        while (j >= k) {
            j -= k;
            k /= 2;
        }
        if (j < k) j += k;
    }
}

void fft(plex y[],int len,int on)
{
    change(y,len);
    for(int h = 2; h <= len; h <<= 1)
    {
        plex wn(cos(-on*2*pi/h),sin(-on*2*pi/h));
        for(int j = 0;j < len;j+=h)
        {
            plex w(1,0);
            for(int k = j;k < j+h/2;k++)
            {
                plex u = y[k];
                plex t = w*y[k+h/2];
                y[k] = u+t;
                y[k+h/2] = u-t;
                w = w*wn;
            }
        }
    }
    if(on == -1)
        for(int i = 0;i < len;i++)
            y[i].x /= len;
}

int n;
int dp[maxn], a[maxn];

void solve (int l, int r) {
    if(l == r){
        dp[l] += a[l];
        dp[l] %= mod;
        return;
    }
    int mid = (l+r) >> 1;
    solve (l, mid);
    int len = 1;
    while (len <= r-l+1) {len <<= 1;}
    for (int i = 0; i < len; i++) {
        x1[i] = x2[i] = plex (0, 0);
    }
    for (int i = l; i <= mid; i++) x1[i-l] = plex (dp[i], 0);
    for (int i = 1; i <= r-l; i++) x2[i-1] = plex (a[i], 0);
    fft (x1, len, 1), fft (x2, len, 1);
    for (int i = 0; i < len; i++) x1[i] = x1[i] * x2[i];
    fft (x1, len, -1);
    for (int i = mid+1; i <= r; i++) {
        dp[i] += (int) (x1[i-l-1].x + 0.5);
        dp[i] %= mod;
    }
    solve (mid+1, r);
}

int main () {
    while (scanf ("%d", &n) == 1 && n) {
        for (int i = 1; i <= n; i++) {
            scanf ("%d", &a[i]);
            a[i] %= mod;
            dp[i] = 0;
        }
        dp[0] = 0;
        solve (1, n);
        printf ("%d\n", dp[n]);
    }
    return 0;
}
```

任意模数的FFT

使用前调用fft_init() 使用的时候 a的下标是[0,n]，b的下标为[0,m]

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <ctime>
using namespace std;
#define rep(i,a,b) for (int i=(a),_ed=(b);i<_ed;i++)
#define per(i,a,b) for (int i=(b)-1,_ed=(a);i>=_ed;i--)
#define upmo(a,b) (((a)=((a)+(b))%mo)<0?(a)+=mo:(a))
typedef long long ll;
typedef double db;
inline ll sqr(ll a){return a*a;}
inline db sqrf(db a){return a*a;}
const int inf=0x3f3f3f3f;
```

```
//const ll inf=0x3f3f3f3f3f3f3f3fll;
const db pi=3.14159265358979323846264338327950288L;
const db eps=1e-6;
//const int mo=0;
//int qp(int a,ll b){int n=1;do{if(b&1)n=1ll*n*a%mo;a=1ll*a*a%mo;}while(b>>=1);return
n;}


// FFT_MAXN = 2^k
// first call fft_init() to precalc FFT_MAXN-th roots
// convo(a,n,b,m,c) a[0..n]*b[0..m] -> c[0..n+m]
typedef double db;
const int FFT_MAXN=262144,N=FFT_MAXN*2;
int mo,n;
struct cp{
    db a,b;
    cp operator+(const cp&y)const{return (cp){a+y.a,b+y.b};}
    cp operator-(const cp&y)const{return (cp){a-y.a,b-y.b};}
    cp operator*(const cp&y)const{return (cp){a*y.a-b*y.b,a*y.b+b*y.a};}
    cp operator!()const{return (cp){a,-b};};
}nw[FFT_MAXN+1];int bitrev[FFT_MAXN];
void dft(cp*a,int n,int flag=1){
    int d=0;while((1<<d)*n!=FFT_MAXN)d++;
    rep(i,0,n)if(i<(bitrev[i]>>d))swap(a[i],a[bitrev[i]>>d]);
    for (int l=2;l<=n;l<<=1){
        int del=FFT_MAXN/l*flag;
        for (int i=0;i<n;i+=l){
            cp *le=a+i,*ri=a+i+(l>>1),*w=flag==1?nw:nw+FFT_MAXN;
            rep(k,0,l>>1){
                cp ne=*ri**w;
                *ri=*le-ne,*le=*le+ne;
                le++,ri++,w+=del;
            }
        }
    }
    if(flag!=1)rep(i,0,n)a[i].a/=n,a[i].b/=n;
}
void fft_init(){
    int L=0;while((1<<L)!=FFT_MAXN)L++;
    bitrev[0]=0;rep(i,1,FFT_MAXN)bitrev[i]=bitrev[i>>1]>>1|((i&1)<<(L-1));
    rep(i,0,FFT_MAXN+1)nw[i]=(cp){(db)cosl(2*pi/FFT_MAXN*i),
(db)sinl(2*pi/FFT_MAXN*i)};
}
void convo(int*a,int n,int*b,int m,int*c){
    rep(i,0,n+m+1)c[i]=0;
    if(n<=100 && m<=100 || min(n,m)<=5){
        static int tmp[FFT_MAXN];
    //  cout<<"fft"<<endl;
    //  for (int i=0;i<n;i++) cout<<a[i]<<" "; cout<<endl;
    //  for (int i=0;i<m;i++) cout<<b[i]<<" "; cout<<endl;
        rep(i,0,n+m+1)tmp[i]=0;
        rep(i,0,n+1)rep(j,0,m+1)upmo(tmp[i+j],1ll*a[i]*b[j]);
        rep(i,0,n+m+1)c[i]=tmp[i];
    //  for (int i=0;i<n+m+1;i++) cout<<c[i]<<" "; cout<<endl;
        return;
    }
    static cp f[FFT_MAXN],g[FFT_MAXN],t[FFT_MAXN];
    int N=2;while(N<=n+m)N<<=1; //cout<<N<<endl;
    rep(i,0,N){
        int aa=i<=n?a[i]:0,bb=i<=m?b[i]:0;upmo(aa,0);upmo(bb,0);
        f[i]=(cp){db(aa>>15),db(aa&32767)};
        g[i]=(cp){db(bb>>15),db(bb&32767)};
    }
    dft(f,N);dft(g,N);
    rep(i,0,N){
        int j=i?N-i:0;
        t[i]=((f[i]+!f[j])*(!g[j]-g[i])+(!f[j]-f[i])*(g[i]+!g[j]))*(cp){0,0.25};
    }
    dft(t,N,-1);
    rep(i,0,n+m+1)upmo(c[i],(ll(t[i].a+0.5))%mo<<15);
    rep(i,0,N){
        int j=i?N-i:0;
        t[i]=(!f[j]-f[i])*(!g[j]-g[i])*(cp){-0.25,0}+(cp){0,0.25}*(f[i]+!f[j])*
(g[i]+!g[j]);
    }
    dft(t,N,-1);
    rep(i,0,n+m+1)upmo(c[i],ll(t[i].a+0.5)+(ll(t[i].b+0.5)%mo<<30));
}
```

关于计数的FFT

`Asia Hong Kong Online Preliminary A`:给出n个数字（$\in[-50000,50000]$）,求有多少(i,j,k)满足$a_i+a_j=a_k$(要求i,j,k互不相同)。

- 0比较麻烦直接拎出来记录个数，其他的加一个50000避免负数直接存起来。然后开一个数组记录每一个数字出现多少次，这个数组和自己做一次卷积即可。答案分成几部分：

1. $i+j=k$: 直接遍历扔掉0的数组，把算完卷积的结果扔进去；
2. $0+0=0$：直接用0的个数可以得出结果；
3. $x+y=0$：直接用卷积中0次项的系数 或者暴力遍历；
4. $0+x=x+0=x$：暴力遍历统计。

```
#define add 50000

struct plex {
    double x, y;
    plex (double _x = 0.0, double _y = 0.0) : x (_x), y (_y) {}
    plex operator + (const plex &a) const {
        return plex (x+a.x, y+a.y);
    }
    plex operator - (const plex &a) const {
        return plex (x-a.x, y-a.y);
    }
    plex operator * (const plex &a) const {
        return plex (x*a.x-y*a.y, x*a.y+y*a.x);
    }
}x[maxn];

void change (plex *y, int len) {
    int i, j, k;
    for(i = 1, j = len / 2; i < len - 1; i++) {
        if (i < j) swap(y[i], y[j]);
        k = len / 2;
        while (j >= k) {
            j -= k;
            k /= 2;
        }
        if (j < k) j += k;
    }
}

void fft(plex y[],int len,int on)
{
    change(y,len);
    for(int h = 2; h <= len; h <<= 1)
    {
        plex wn(cos(-on*2*pi/h),sin(-on*2*pi/h));
        for(int j = 0;j < len;j+=h)
        {
            plex w(1,0);
            for(int k = j;k < j+h/2;k++)
            {
                plex u = y[k];
                plex t = w*y[k+h/2];
                y[k] = u+t;
                y[k+h/2] = u-t;
                w = w*wn;
            }
        }
    }
    if(on == -1)
        for(int i = 0;i < len;i++)
            y[i].x /= len;
}

int n, cnt, zero;
int a[maxn], num[maxn];

long long solve () {
    long long ans = 0;
    int len = 1;
    while (len < 200000) {
        len <<= 1;
    }
    for (int i = 0; i < len; i++) {
        x[i] = plex (num[i], 0);
    }
    fft (x, len, 1);
    for (int i = 0; i < len; i++) {
        x[i] = x[i]*x[i];
    }
    fft (x, len, -1);

    for (int i = 1; i <= cnt; i++) {//i+j=k
        long long tmp = (long long) (x[a[i]+add*2].x + 0.5);
        ans += tmp;
    }
    for (int i = 1; i <= cnt; i++) {//减去i+i=j
        if (a[i]%2 == 0)
            ans -= num[a[i]/2+add];
    }

    if (zero >= 3) ans += 1LL*zero*(zero-1)*(zero-2);//0+0=0
    for (int i = 0; i <= add*2; i++) {//x+0=y y+0=x
        if (num[i] > 1) {
            ans += 1LL*2*num[i]*(num[i]-1)*zero;
        }
    }
    ans += 1LL*(long long)(x[2*add].x+0.5)*zero;//x+y=0

    return ans;
}

int main () {
    scanf ("%d", &n);
    zero = cnt = 0;
    memset (num, 0, sizeof num);
    for (int i = 1; i <= n; i++) {
        scanf ("%d", &a[++cnt]);
        if (a[cnt] == 0) {
            zero++;
            cnt--;
            continue;
        }
        num[a[cnt]+add]++;
    }
    long long ans = solve ();
    printf ("%lld\n", ans);
    return 0;
}
```

NTT

- 把FFT中的单位负根用数论中的原根代替，其他代码完全一致，可以避免精度误差。

```
//G表示模数的原根
long long x1[maxn], x2[maxn];

void change (long long y[], int len) {
    for(int i = 1, j = len / 2; i < len - 1; i++) {
        if(i < j) swap(y[i], y[j]);
        int k = len / 2;
        while(j >= k) {
            j -= k;
            k /= 2;
        }
        if(j < k) j += k;
    }
}
void ntt(long long y[], int len, int on) {
    change (y, len);
    for(int h = 2; h <= len; h <<= 1) {
        long long wn = qpow(G, (mod-1)/h);
        if(on == -1) wn = qpow(wn, mod-2);
        for(int j = 0; j < len; j += h) {
            long long w = 1;
            for(int k = j; k < j + h / 2; k++) {
                long long u = y[k];
                long long t = w * y[k + h / 2] % mod;
                y[k] = (u + t) % mod;
                y[k+h/2] = (u - t + mod) % mod;
                w = w * wn % mod;
            }
        }
    }
    if(on == -1) {
        long long t = qpow (len, mod - 2);
        for(int i = 0; i < len; i++)
            y[i] = y[i] * t % mod;
    }
}
```

`HDU 5552` :n个点的有环连通图计数，每条边有m种颜色可以染。 设：

1. f[n]:n个点的连通图数；
2. g[n]:n个点的图数，$g[n]=(m+1)^{\frac{n*(n-1)}{2}}$
3. h[n]:n个点的树数，$h[n]=n^{n-2}*m^{n-1}$

$f[n]=g[n]-(n-1)!\sum_{i=1}^{n-1}(i-1)f[i]*(n-i)g[n-i];ans=f[n]-h[n]$。

```
#define mod 152076289
#define G 106
#define maxn 40005

long long x1[maxn], x2[maxn];
```

```cpp
long long n, m;
long long f[maxn], g[maxn], h[maxn];
//n个点的连通图 n个点的图 n个点的树
long long c[maxn], fac[maxn], rev[maxn];

void solve (int l, int r) {
    if (l == r) {
        f[l] += g[l];
        f[l] %= mod;
        return;
    }
    int mid = (l+r) >> 1;
    solve (l, mid);
    int len = 1;
    while (len <= r-l+1) {len <<= 1;}
    for (int i = 0; i < len; i++) {
        x1[i] = x2[i] = 0;
    }
    for (int i = l; i <= mid; i++) {
        x1[i-l] = f[i]*rev[i-1]%mod;
    }
    for (int i = 1; i <= r-l; i++) {
        x2[i-1] = g[i]*rev[i]%mod;
    }
    ntt (x1, len, 1), ntt (x2, len, 1);
    for (int i = 0; i < len; i++) x1[i] = x1[i] * x2[i] % mod;
    ntt (x1, len, -1);
    for (int i = mid+1; i <= r; i++) {
        f[i] -= x1[i-l-1] %mod *fac[i-1] %mod;
        (f[i] += mod) %= mod;
    }
    solve (mid+1, r);
}

int main () {
    int t, kase = 0;
    fac[0] = 1;
    for (int i = 1; i < maxn; i++) fac[i] = fac[i-1] * i % mod;
    rev[maxn-1] = qpow (fac[maxn-1], mod - 2);
    for (int i = maxn-2; i >= 0; i--) rev[i] = rev[i+1] * (i+1) % mod;
    scanf ("%d", &t);
    while (t--) {
        scanf ("%lld%lld", &n, &m);
        memset (f, 0, sizeof f);
        c[0] = 1;
        for (int i = 1; i < n; i++) {
            c[i] = c[i-1]*(n-1-i+1)%mod*rev[i]%mod;
        }
        for (int i = 1; i <= n; i++) {
            g[i] = qpow (1+m, i*(i-1)/2);
        }
        solve (1, n);
        long long ans = f[n];
        long long tmp = qpow (n, n-2) * qpow (m, n-1) % mod;
        ans = (ans - tmp + mod) % mod;
        printf ("Case #%d: %lld\n", ++kase, ans);
    }
    return 0;
}
```

`HDU 5829` :定义f(k)为集合S中$\sum$所有子集前k大数的和，求所有的k($1\le k\le n$).

从大到小排序，考虑第k个数的贡献f_k，那么有：
$f_k=\sum_{i=k}^{n}C_{i-1}^{k-1}*A_i*2^{n-i}=\frac{1}{2^k*(k-1)!}\sum_{i=0}^{n-k}\frac{(i+k-1)!}{i!}*2^{n-i}*A_{i+k}$ 然后令：
$g(i)=\frac{2^i}{(n-i)!},h(i)=(i-1)!*A_i$, 所以$f_k=\frac{1}{2^k*(k-1)!}\sum_{i=0}^{n-i}g(n-i)*h(i+k)$卷积就很显然了。

`2016北京online F` : 求$min\left\{\sum_0^{n-1}(A_i-B_{(i+k)\ mod\ n})^2\|k=0,1\ldots n-1\right\}$.

- 把A数组逆置过来倍增一倍的长度，显然结果就是卷积的某一项。结果比较比，所以NTT时取两个10^9左右的模数，然后用CRT合并。

```
#define maxn 1000005
#define mod1 1004535809LL
#define mod2 469762049LL
#define G 3

long long x1[maxn], x2[maxn], x3[maxn], x4[maxn];

long long mul (long long a, long long b, long long mod) {
    if (b == 0)
        return 0;
    long long ans = mul (a, b>>1, mod);
    ans = ans*2%mod;
    if (b&1) ans += a, ans %= mod;
    return ans;
}


long long qpow (long long a, long long b, long long mod) {
    long long ret=1;
    while (b) {
        if (b&1) ret = (ret*a)%mod;
        a = (a*a)%mod;
        b >>= 1;
    }
    return ret;
}

void gcd (long long a, long long b, long long &d, long long &x, long long &y) {
    if (!b) {
        d = a;
        x = 1;
        y = 0;
    }
    else {
        gcd (b, a%b, d, y, x);
        y -= x*(a/b);
    }
}

long long aa[11], mm[11];
long long china (int n, long long *a, long long *m) {
    long long M = 1, d, y, x = 0;
    for (int i = 0; i < n; i++) M *= m[i];
    for (int i = 0; i< n; i++) {
        long long w = M/m[i];
        gcd (m[i], w, d, d, y);
        x = (x+ mul (mul (y, w, M), a[i], M)) % M;
    }
    return (x+M) % M;
}

void change (long long y[], int len) {
    for(int i = 1, j = len / 2; i < len - 1; i++) {
        if(i < j) swap(y[i], y[j]);
        int k = len / 2;
        while(j >= k) {
            j -= k;
            k /= 2;
        }
        if(j < k) j += k;
    }
}
void fft (long long y[], int len, int on, long long mod) {
    change (y, len);
    for(int h = 2; h <= len; h <<= 1) {
        long long wn = qpow(G, (mod-1)/h, mod);
        if(on == -1) wn = qpow(wn, mod-2, mod);
        for(int j = 0; j < len; j += h) {
            long long w = 1;
            for(int k = j; k < j + h / 2; k++) {
                long long u = y[k];
                long long t = w * y[k + h / 2] % mod;
                y[k] = (u + t) % mod;
                y[k+h/2] = (u - t + mod) % mod;
                w = w * wn % mod;
            }
        }
    }
    if(on == -1) {
        long long t = qpow (len, mod - 2, mod);
        for(int i = 0; i < len; i++)
            y[i] = y[i] * t % mod;
    }
}

int n;
long long a[maxn], b[maxn];

void solve () {
    for (int i = 0; i < n/2; i++) {
        swap (b[i], b[n-i-1]);
    }
    for (int i = n; i < 2*n; i++) {
        b[i] = b[i-n];
    }
    int len = 1;
    while (len < 2*n) len <<= 1;
    for (int i = 0; i < n; i++) {
        x1[i] = x3[i] = a[i];
    }
    for (int i = n; i < len; i++) {
        x1[i] = x3[i] = 0;
    }
    for (int i = 0; i < 2*n; i++) {
        x2[i] = x4[i] = b[i];
    }
    for (int i = 2*n; i < len; i++) {
        x2[i] = x4[i] = 0;
    }
    fft (x1, len, 1, mod1);
    fft (x2, len, 1, mod1);
    fft (x3, len, 1, mod2);
    fft (x4, len, 1, mod2);
    for (int i = 0; i < len; i++) {
        x1[i] = x1[i]*x2[i]%mod1;
        x3[i] = x3[i]*x4[i]%mod2;
    }
    fft (x1, len, -1, mod1);
    fft (x3, len, -1, mod2);
    long long ans = 0;
    for (int i = n-1; i <= 2*n-2; i++) {
        mm[0] = mod1, mm[1] = mod2;
        aa[0] = x1[i], aa[1] = x3[i];
        long long tmp = china (2, aa, mm);
        ans = max (ans, tmp);
    }
    long long tot = 0;
    for (int i = 0; i < n; i++) {
        tot += a[i]*a[i] + b[i]*b[i];
    }
    printf ("%lld\n", tot-2*ans);
}


int main () {
    int t;
    scanf ("%d", &t);
    while (t--) {
        scanf ("%d", &n);
        for (int i = 0; i < n; i++) {
            scanf ("%lld", &a[i]);
        }
        for (int j = 0; j < n; j++) {
            scanf ("%lld", &b[j]);
        }
        solve ();
    }
    return 0;
}
```

FWT

解决逻辑卷积。

```
#define ll long long
#define mod 1000000007
//计算逻辑卷积和 （正常卷积是i+j=k 这个可以算i^j=k 等等）
int rev=(mod+1)>>1;
//rev: 2对于mod的逆元
void FWT(int a[],int n)
{
    for(int d=1;d<n;d<<=1)
        for(int m=d<<1,i=0;i<n;i+=m)
            for(int j=0;j<d;j++)
            {
                int x=a[i+j],y=a[i+j+d];
                a[i+j]=(x+y)%mod,a[i+j+d]=(x-y+mod)%mod;
                //xor:a[i+j]=x+y,a[i+j+d]=(x-y+mod)%mod;
                //and:a[i+j]=x+y;
                //or:a[i+j+d]=x+y;
            }
}

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

一般数据结构

STL相关

哈希表

- STL版本（需要c++11）

用法和map，set一样

```
#include <unordered_set>
#include <unordered_map>
```

- 手动实现版本

```
const int MAXN = 400005; //最多出现多少个不同的值
const int seed = 1000007; //选择的模数

struct hashmap {
    long long a[MAXN];
    int head[seed], next[MAXN], size;
    void init () {  //初始化哈希表
            memset (head, -1, sizeof head);
        size = 0;
    }
    bool find (long long val) {  //查找一个元素是否在哈希表内
        int tmp = (val%seed + seed)%seed;
        for (int i = head[tmp]; i != -1; i = next[i]) {
            if (val == a[i]) return true;
        }
        return false;
    }
    void add (long long val) {  //添加元素到哈希表中
        int tmp = (val%seed+seed)%seed;
        if (find (val)) return;
        a[size] = val;
```

```
            next[size] = head[tmp];
            head[tmp] = size++;
        }
    }table;
```

单调栈

`NEU 1643`：定义f(l,r)表示a[l]到a[r]中的最大值，每一组询问(l,r)求$\sum_{i=l}^{r}\sum_{j=i}^{r} f(i,r)$

单调栈维护每个数以它为最大最左（右）延伸到的下标即可。

```
struct node {
    long long pos, num;
} a[maxn];
long long l[maxn], r[maxn], top, n;
node st[maxn]; //单调栈
long long ans;

int main () {
    while (scanf ("%lld", &n) == 1) {
        ans = 0;
        for (int i = 1; i <= n; i++) {
            l[i] = i;
            r[i] = i;
        }
        top = 0;
        for (int i = 1; i <= n; i++) {
            scanf ("%lld", &a[i].num);
            a[i].pos = i;
        }
        for (int i = 1; i <= n; i++) {
            if (top == 0) { //空
                st[++top] = a[i];
            }
            else if (a[i].num <= st[top].num) {
                st[++top] = a[i];
            }
            else {
                while (top != 0 && st[top].num < a[i].num) {
                    node now = st[top];
                    r[now.pos] = i-1;
                    top--;
                }
                st[++top] = a[i];
            }
        }
        if (top != 0) {
            int rr = st[top].pos;
            while (top != 0) {
                r[st[top].pos] = rr;
                top--;
            }
        }
        top = 0;
        for (int i = n; i >= 1; i--) {
            if (top == 0) { //空
                st[++top] = a[i];
            }
            else if (a[i].num < st[top].num) {
                st[++top] = a[i];
            }
            else {
                while (top != 0 && st[top].num <= a[i].num) {
                    node now = st[top];
                    l[now.pos] = i+1;
                    top--;
                }
                st[++top] = a[i];
            }
        }
        if (top != 0) {
            int ll = st[top].pos;
            while (top != 0) {
                l[st[top].pos] = ll;
                top--;
            }
        }
        for (long long i = 1; i <= n; i++)
            ans += (i-l[i]+1)*(r[i]-i+1)*a[i].num;
        printf ("%lld\n", ans);
    }
    return 0;
}
```

- 解决最大全0矩阵，全0矩阵计数等问题。

```
//全0矩阵计数
pii s[maxn]; int p;
int h[maxn][maxn];
int cal () {
    for (int i = 1; i <= m; i++) h[1][i] = (mp[1][i] == 0);
    for (int i = 2; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (mp[i][j] == 1) h[i][j] = 0;
            else h[i][j] = h[i-1][j]+(mp[i][j] == 0);
        }
    }
    int ans = 0;
    for (int i = 1; i <= n; i++) {
        int cnt = 0;
        p = 0;
        for (int j = 1; j <= m; j++) {
            pii tmp = make_pair (h[i][j], 1);
            while (p > 0 && s[p-1].fi >= tmp.fi) {
                pii u = s[p-1]; p--;
                cnt -= u.fi*u.se;
                tmp.se += u.se;
            }
            s[p++] = tmp;
            cnt += tmp.fi*tmp.se;
            ans += cnt;
        }
    }
    return ans;
}
```

并查集

查询并查集根

```
int fa[maxn];
int Find (int x) {
    return x == fa[x] ? fa[x] : fa[x] = Find (fa[x]);
}
```

种类并查集

`neuoj 180`:给出任意两个人的关系，询问是否有矛盾。

```
int find (int x) {
    if (x == fa[x]) return x;
    int cur = fa[x];
    fa[x] = find (fa[x]);
    ra[x] = (ra[cur] + ra[x]) % 2;
    return fa[x];
}

void union_set (int a, int b) {
    int p1 = find (a);
    int p2 = find (b);
    fa[p2] = p1;
    ra[p2] = (ra[a] + 1 - ra[b]) % 2;
}

int main() {
    int t, kase = 0;
    scanf ("%d", &t);
    while (t--) {
        if (kase) printf ("\n");
        printf ("Test case #%d:\n", ++kase);
        scanf ("%d%d", &n, &m);
        for (int i = 1; i <= n; i++) {
            fa[i] = i;
            ra[i] = 0;
        }
        bool ok = 1;
        for (int i = 0; i < m; i++) {
            scanf ("%d%d", &a, &b);
            int p1 = find (a);
            int p2 = find (b);
            if (p1 == p2) {
                if (ra[a] == ra[b]) ok = 0;
                else continue;
            }
            else {
                union_set (a, b);
            }
        }
        if (ok) printf ("Nothing special.\n");
        else printf ("Something wrong!\n");
    }
    return 0;
}
```

```
#include<cstdio>
#include<map>
#include <iostream>
#define N 200500
using namespace std;
int f[N],Rank[N], n, m,q;
bool flag;

inline void init(){
    flag=false;
    for(int i=0; i<=n; ++i)
        f[i]=i, Rank[i]=0;
}
int find(int x){
    if(x==f[x])return f[x];
    int t=find(f[x]);
    Rank[x] = (Rank[f[x]]+Rank[x])&1;
    f[x]=t;
    return f[x];
}
void Union(int x, int y){
    int a=find(x), b=find(y);
    if (flag) return ;
    if (a==b)
    {
        if (Rank[x]==Rank[y])
        {
            flag=1;
            return ;
        }
    }
    f[a]=b;
    Rank[a] = (Rank[x]+Rank[y]+1)&1;
}
int main()
{
    int t;
    cin>>t;
    for (int h=1;h<=t;h++)
    {
        scanf("%d %d",&n,&m);
        init();
        for(int i=0;i<m;i++)
        {
            int u,v;
            scanf("%d %d",&u,&v);
            Union(u,v);
        }
        printf("Scenario #%d:\n",h);
        if (flag)
        {
            printf("Suspicious bugs found!\n\n");
        }
        else
        {
            printf("No suspicious bugs found!\n\n");
        }
    }
    return 0;
}
```

`POJ 1182`:

RMQ

- 解决区间最大，区间最小，区间gcd。初始化O(nlgn)，查询O(1)。

```
//注意下标[0,n-1]
void rmq_init () {
    for (int i = 0; i < n; i++) dp[i][0] = a[i];
    for (int j = 1; (1<<j) <= n; j++) {
        for (int i = 0; i+(1<<j)-1 < n; i++) {
            dp[i][j] = __gcd (dp[i][j-1], dp[i+(1<<(j-1))][j-1]);
        }
    }
}
```

```
int rmq (int l, int r) {
    int k = 0;
    while ((1<<(k+1)) <= r-l+1) k++;
    return __gcd (dp[l][k], dp[r-(1<<k)+1][k]);
}
```

- 二维RMQ

```
//二维坐标从(1,1)到(n,m)
int n, m;
int val[maxn][maxn];
int dp[maxn][maxn][9][9];
void rmq_init ()
{
    for(int row = 1; row <= n; row++)
        for(int col = 1; col <=m; col++)
            dp[row][col][0][0] = val[row][col];
    int mx = log(double(n)) / log(2.0);
    int my = log(double(m)) / log(2.0);
    for(int i=0; i<= mx; i++)
    {
        for(int j = 0; j<=my; j++)
        {
            if(i == 0 && j ==0) continue;
            for(int row = 1; row+(1<<i)-1 <= n; row++)
            {
                for(int col = 1; col+(1<<j)-1 <= m; col++)
                {
                    if(i == 0)//y轴二分
                        dp[row][col][i][j]=max(dp[row][col][i][j-1],dp[row][col+
(1<<(j-1))][i][j-1]);
                    else//x轴二分
                        dp[row][col][i][j]=max(dp[row][col][i-1][j],dp[row+(1<<(i-
1))][col][i-1][j]);
                }
            }
        }
    }
}
int rmq (int x1,int x2,int y1,int y2)
{
    int kx = log(double(x2-x1+1)) / log(2.0);
    int ky = log(double(y2-y1+1)) / log(2.0);
    int m1 = dp[x1][y1][kx][ky];
    int m2 = dp[x2-(1<<kx)+1][y1][kx][ky];
    int m3 = dp[x1][y2-(1<<ky)+1][kx][ky];
    int m4 = dp[x2-(1<<kx)+1][y2-(1<<ky)+1][kx][ky];
    return max( max(m1,m2) , max(m3,m4));
}
```

树状数组

- lowbit函数：一个数二进制右边第一个1代表的权重，如lowbit(6)=2.

```
int lowbit (int x) {
        return x&(-x);
}
```

- 一维写法:

```
void add (int x, int num) {
    for (int i = x; i; i -= lowbit (i)) c[i] += num;
}

int sum (int x) {
    int ans = 0;
    for (int i = x; i < maxn; i += lowbit (i)) ans += c[i];
    return ans;
}
```

- 二维写法:

```
void add (int x, int y, int num) {
    for (int i = x; i; i -= lowbit (i)) {
        for (int j = y; j; j -= lowbit (j))
            c[i][j] += num;
    }
}

int sum (int x, int y) {
    int ans = 0;
    for (int i = x; i < maxn; i += lowbit (i)) {
        for (int j = y; j < maxn; j += lowbit (j))
            ans += c[i][j];
    }
    return ans;
}
```

子矩阵更新单点求和

```
//(x1,y1)是左下角的点，(x2,y2)是右上角的点
add (x1, y1, num); add (x2+1, y2+1, num); add (x1, y2+1, num); add (x2+1, y1, num);
//查询(x,y)的值
sum (x, y);
```

区间中出现的数字种数

`SPOJ DQUUERY`：询问区间内出现的数字有多少种。 变形：给出一系列区间，每次询问一个区间中有多少给出的区间(HDU 5775)。

都是按照离线存储询问，然后一维排序一维扫描，扫描的同时用树状数组维护。

```
int a[maxn];
int n;
vector <int> num;
map <int , int> gg;
struct node {
    int l, r, id;
    bool operator < (const node &a) const {
        return r < a.r;
    }
}op[maxn];
int ans[maxn], pos[maxn];

void init () {
    int cnt = 0;
    sort (num.begin (), num.end ());
    for (int i = 0; i < n; i++) {
        if (!i || num[i] != num[i-1]) {
            gg[num[i]] = ++cnt;
        }
    }
    for (int i = 1; i <= n; i++) {
        a[i] = gg[a[i]];
    }
}

int c[maxn];

int lowbit (int x) {
    return x&(-x);
}

void add (int pos, int num) {
    for (int i = pos; i <= n; i += lowbit (i))
        c[i] += num;
}

int sum (int pos) {
    int ans = 0;
    for (int i = pos; i > 0; i -= lowbit (i))
        ans += c[i];
    return ans;
}

int main () {
    while (scanf ("%d", &n) == 1) {
        gg.clear ();
        num.clear ();
        for (int i = 1; i <= n; i++) {
            scanf ("%d", &a[i]);
            num.push_back (a[i]);
        }
        init ();
        int q;
        scanf ("%d", &q);
        for (int i = 1; i <= q; i++) {
            scanf ("%d%d", &op[i].l, &op[i].r);
            op[i].id = i;
        }
        sort (op+1, op+1+q);
        memset (c, 0, sizeof c);
        memset (pos, -1, sizeof pos);
        int pre = 0;
        for (int i = 1; i <= q; i++) {
            for (; pre+1 <= op[i].r;) {
                pre++;
                add (pre, 1);
                if (pos[a[pre]] != -1) {
                    add (pos[a[pre]], -1);
                }
                pos[a[pre]] = pre;
            }
            ans[op[i].id] = sum (op[i].r) - sum (op[i].l-1);
        }
        for (int i = 1; i <= q; i++) {
            printf ("%d\n", ans[i]);
        }
    }
    return 0;
}
```

线段树

经常写挫的地方： **lazy标记在初始化的时候都要初始化！！！！！！**

树链剖分

- 点权 `HDU 3966`:树上路径修改，单点查询。用树状数组维护树链。

```
int n, m, q;
struct node {
    int u, v, next;
}edge[maxm];
int head[maxn], cnt, a[maxn];

void add_edge (int u, int v) {
    edge[cnt] = (node) {u, v, head[u]};
    head[u] = cnt++;
}
```

```
int son[maxn], sz[maxn], top[maxn], deep[maxn], fa[maxn];
int tot, p[maxn];
void dfs (int u, int father, int d) {
    fa[u] = father;
    sz[u] = 1;
    deep[u] = d;
    for (int i = head[u]; i+1; i = edge[i].next) {
        int v = edge[i].v; if (v == father) continue;
        dfs (v, u, d+1);
        sz[u] += sz[v];
        if (son[u] == -1 || sz[v] > sz[son[u]]) {
            son[u] = v;
        }
    }
}

void get_pos (int u, int fa, int sp) {
    top[u] = sp;
    p[u] = ++tot;
    if (son[u] == -1) return ;
    get_pos (son[u], u, sp);
    for (int i = head[u]; i+1; i = edge[i].next) {
        int v = edge[i].v;
        if (v == fa || v == son[u]) continue;
        get_pos (v, u, v);
    }
}

long long c[maxn];
void add (int x, int num) {
    for (int i = x; i < maxn; i += lowbit (i)) {
        c[i] += num;
    }
}

long long query (int x) {
    long long ans = 0;
    for (int i = x; i > 0; i -= lowbit (i)) {
        ans += c[i];
    }
    return ans;
}

void update (int u, int v, int num) {
    int f1 = top[u], f2 = top[v];
    while (f1 != f2) {
        if (deep[f1] < deep[f2]) {
            swap (u, v); swap (f1, f2);
        }
        add (p[f1], num);
        add (p[u]+1, -num);
        u = fa[f1], f1 = top[u];
    }
    if (deep[u] < deep[v]) {
        swap (u, v);
    }
    add (p[v], num);
    add (p[u]+1, -num);
}

void solve () {
    char op[2];
    memset (c, 0, (n+5)*sizeof c[0]);
    for (int i = 1; i <= n; i++) {
        add (p[i], a[i]);
        add (p[i]+1, -a[i]);
    }
    for (int i = 0; i < q; i++) {
        scanf ("%s", op);
        if (op[0] == 'I') {
            int u, v, num; scanf ("%d%d%d", &u, &v, &num);
            update (u, v, num);
        }
        else if (op[0] == 'D') {
            int u, v, num; scanf ("%d%d%d", &u, &v, &num);
            update (u, v, -num);
        }
        else {
            int u; scan (u);
            long long ans = query (p[u]);
            printf ("%lld\n", ans);
        }
    }
}

int main () {
    while (scanf ("%d%d%d", &n, &m, &q) == 3) {
        for (int i = 1; i <= n; i++) {
            scan (a[i]);
        }
        memset (head, -1, (n+5)*sizeof head[0]); cnt = 0;
        for (int i = 1; i < n; i++) {
            int u, v; scan (u); scan (v);
            add_edge (u, v);
            add_edge (v, u);
        }
        memset (son, -1, (n+5)*sizeof son[0]); tot = 0;
        dfs (1, 0, 0);
        get_pos (1, 0, 1);
        solve ();
    }
    return 0;
}
```

- 边权

`BZOJ 1984`:树链修改 树链加 树链求最大值。

```
int pos, n;
int son[maxn], num[maxn], top[maxn]; //重儿子 子树size 重链的顶
int p[maxn], fp[maxn], fa[maxn], deep[maxn]; //每个点对应的线段树的点 线段树的点对应的树上点
每个点的父亲 每个点的深度
int wfa[maxn]; //父亲连向它的边的权值
void init () {
    memset (son, -1, (n+5)*sizeof son[0]); pos = 0;
    wfa[0] = 0; //根的父亲没有意义
}

void dfs1 (int u, int pre, int d) {
    deep[u] = d;
    fa[u] = pre;
    num[u] = 1;
    for (int i = head[u]; i+1; i = edge[i].next) {
        int v = edge[i].v; if (v == pre) continue;
        dfs1 (v, u, d+1);
        num[u] += num[v];
        wfa[v] = edge[i].w;
        if (son[u] == -1 || num[son[u]] < num[v]) {
            son[u] = v;
        }
    }
}

void dfs2 (int u, int sp) {
    top[u] = sp;
    p[u] = ++pos;
    fp[p[u]] = u;
    if (son[u] == -1) return ;
    dfs2 (son[u], sp);
    for (int i = head[u]; i+1; i = edge[i].next) {
        int v = edge[i].v;
        if (v != fa[u] && v != son[u]) {
            dfs2 (v, v);
        }
    }
}

struct Node {
    int num; //区间改变成一个值 如果是-INF表示没有改变
    int add; //区间增加一个值
    int Max; //区间的最大值
}tree[maxn<<2];

void build_tree (int l, int r, int c) {
    tree[c].num = -INF; tree[c].add = 0;
    if (l == r) {
        tree[c].Max = wfa[fp[l]];
        return ;
    }
    int mid = (l+r)>>1;
    build_tree (lson);
    build_tree (rson);
    tree[c].Max = max (tree[pl].Max, tree[pr].Max);
}

void push_down (int c) {
    if (tree[c].num != -INF) {
        tree[pl].num = tree[pr].num = tree[c].num;
        tree[pl].add = tree[pr].add = 0;
        tree[pl].Max = tree[pr].Max = tree[c].num;
        tree[c].num = -INF;
    }
    if (tree[c].add != 0) {
        tree[pl].add += tree[c].add; tree[pr].add += tree[c].add;
        tree[pl].Max += tree[c].add; tree[pr].Max += tree[c].add;
        tree[c].add = 0;
    }
}

void update (int l, int r, int c, int x, int y, int val, int &op) { //op=0:cover
op=1:add
    if (l != r) push_down (c);
    if (l == x && y == r) {
        if (op == 0) {
            tree[c].num = tree[c].Max = val;
            tree[c].add = 0;
        }
        else {
            tree[c].Max += val;
            tree[c].add += val;
        }
        return ;
    }
    int mid = (l+r)>>1;
    if (mid >= y) update (lson, x, y, val, op);
    else if (mid < x) update (rson, x, y, val, op);
    else {
        update (lson, x, mid, val, op);
        update (rson, mid+1, y, val, op);
    }
    tree[c].Max = max (tree[pl].Max, tree[pr].Max);
}
```

```
int query (int l, int r, int c, int x, int y) {
    if (l != r) push_down (c);
    if (l == x && y == r) {
        return tree[c].Max;
    }
    int mid = (l+r)>>1;
    if (mid >= y) return query (lson, x, y);
    else if (mid < x) return query (rson, x, y);
    else return max (query (lson, x, mid), query (rson, mid+1, y));
}

void solve (int u, int v, int op, int val) { //op=0:区间修改 op=1:区间加 op=2:区间最大值
    int f1 = top[u], f2 = top[v];
    int res = 0;
    while (f1 != f2) {
        if (deep[f1] < deep[f2]) {
            swap (u, v); swap (f1, f2);
        }
        if (op == 2) {
            get_max (res, query (1, pos, 1, p[f1], p[u]));
        }
        else update (1, pos, 1, p[f1], p[u], val, op);
        u = fa[f1], f1 = top[u];
    }
    if (u == v) {
        if (op == 2) printf ("%d\n", res);
        return ;
    }
    if (deep[u] > deep[v]) swap (u, v);
    if (op == 2) {
        printf ("%d\n", max (res, query (1, pos, 1, p[son[u]], p[v])));
    }
    else {
        update (1, pos, 1, p[son[u]], p[v], val, op);
    }
}

int main () {
    scanf ("%d", &n);
    memset (head, -1, (n+5)*sizeof head[0]); cnt = 0;
    for (int i = 1; i < n; i++) {
        int u, v, w; scanf ("%d%d%d", &u, &v, &w);
        add_edge (u, v, w);
        add_edge (v, u, w);
    }
    init ();
    dfs1 (1, 0, 0);
    dfs2 (1, 1);
    build_tree (1, pos, 1);
    char op[11];
    while (scanf ("%s", op) == 1) {
        int u, v, val;
        if (strcmp (op, "Stop") == 0) break;
        if (strcmp (op, "Max") == 0) {
            scanf ("%d%d", &u, &v);
            solve (u, v, 2, 0);
        }
        else if (strcmp (op, "Cover") == 0) {
            scanf ("%d%d%d", &u, &v, &val);
            solve (u, v, 0, val);
        }
        else if (strcmp (op, "Add") == 0) {
            scanf ("%d%d%d", &u, &v, &val);
            solve (u, v, 1, val);
        }
        else if (strcmp (op, "Change") == 0) {
            scanf ("%d%d", &u, &val); u--;
            solve (edge[u<<1].u, edge[u<<1].v, 0, val);
        }
    }
    return 0;
}
```

最近公共祖先

- 离线处理所有的询问,O(m+q)

```
//注意先调用init初始化
int n, m;
struct node {
    int v, next;
}edge[maxm];
int cnt, head[maxn];
int deep[maxn];//深度
struct Q {//询问
    int u, v;
    int lca;
    int next;
}qu[maxm];
int cnt2, head2[maxn];
int p[maxn], fa[maxn];//树上的父亲 并查集根
bool vis[maxn];

#define find Find
int find (int x) {
    return p[x] == x ? p[x] : p[x] = find (p[x]);
}

void add_edge (int u, int v) {
    edge[cnt].v = v, edge[cnt].next = head[u], head[u] = cnt++;
}

void add_query (int u, int v) {
    qu[cnt2].u = u, qu[cnt2].v = v, qu[cnt2].next = head2[u], head2[u] = cnt2++;
}

void LCA (int u, int father, int d) {
    deep[u] = d;
    p[u] = u;
    fa[u] = father;
    vis[u] = 1;
    for (int i = head[u]; i != -1; i = edge[i].next) {
        int v = edge[i].v;
        if (vis[v]) continue;
        LCA (v, u, d+1);
        p[v] = u;
    }
    for (int i = head2[u]; i != -1; i = qu[i].next) {
        int v = qu[i].v;
        if (vis[v]) {
            qu[i^1].lca = qu[i].lca = find (v);
        }
    }
}

void init () {
    for (int i = 0; i <= n; i++) fa[i] = i;
    memset (head, -1, sizeof head);
    memset (vis, 0, sizeof vis);
    cnt = 0;
}
```

- 倍增,O(nlgn)预处理,O(lgn)单次查询

```
//注意先调用init函数初始化
const int DEG = 20;//大约是lgn
struct Edge {
    int to,next, w;
}edge[maxn*2];
int head[maxn], tot;
int n, m, q, root;
void add_edge(int u,int v, int w) {
    edge[tot].to = v;
    edge[tot].w = w;
    edge[tot].next = head[u];
    head[u] = tot++;
}
void init () {
    tot = 0;
    memset (head, -1, sizeof head);
}
int fa[maxn][DEG];//fa[i][j]表示结点i的第2^j个祖先
int deg[maxn];//深度数组
void BFS (int root) {
    queue <int> que;
    deg[root] = 0;
    fa[root][0] = root;
    que.push(root);
    while (!que.empty ()) {
        int tmp = que.front();
        que.pop();
        for (int i = 1; i < DEG; i++)
        fa[tmp][i] = fa[fa[tmp][i-1]][i-1];
        for(int i = head[tmp]; i != -1; i = edge[i].next) {
            int v = edge[i].to;
            if (v == fa[tmp][0]) continue;
            deg[v] = deg[tmp] + 1;
            fa[v][0] = tmp;
            que.push(v);
        }
    }
}
int LCA (int u,int v) {
    if (deg[u] > deg[v]) swap (u, v);
    int hu = deg[u], hv = deg[v];
    int tu = u, tv = v;
    for(int det = hv-hu, i = 0; det; det >>= 1, i++)
        if (det&1)
            tv = fa[tv][i];
        if (tu == tv)
            return tu;
    for(int i = DEG-1; i >= 0; i--) {
        if (fa[tu][i] == fa[tv][i])
            continue;
        tu = fa[tu][i];
        tv = fa[tv][i];
    }
    return fa[tu][0];
}
```

笛卡尔树

两个值key value 从key看 ，整个树是二叉搜索树 从value看，整个树是个堆

```
//根据长度为n的序列构建笛卡尔树，key为下标，value为a[i]   1<=i<=n
void build() {
    int top=0;
    for(int i=1;i<=n;i++)
        l[i]=0,r[i]=0,vis[i]=0;
    for(int i=1;i<=n;i++){
        int k=top;
        while(k>0&&a[stk[k-1]]<a[i])--k;
        //<号是大根堆,>是小根堆
        if (k) r[stk[k-1]]=i;
        if (k<top) l[i]=stk[k];
        stk[k++]=i;
        top=k;
    }
    for(int i=1;i<=n;i++)
        vis[l[i]]=vis[r[i]]=1;
    int rt=0;
    for(int i=1;i<=n;i++)
        if (vis[i]==0) rt=i;
    dfs(rt);
}
```

平衡树

treap树

`codeforces 785E`:动态逆序对，查询交换两个数以后的逆序对数。

树状数组套平衡树即可。

```
struct node *null; //安全节点
struct node {
    node *ch[2];
    int v, r, s;
    int cmp(int x) const {
        if(x == v) return -1;
        return x < v? 0: 1;
    }
    void maintain() {
        s = ch[0]->s + ch[1]->s + 1;
    }
}nodes[maxn*30], *root[maxn];
int cnt;

void rot(node *&o, int d) { //旋转，d为0表示左旋，为1表示右旋
    node *p = o->ch[d^1]; o->ch[d^1] = p->ch[d]; p->ch[d] = o;
    o->maintain(); p->maintain(); o = p;
}

void ins(node *&o, int x) { //插入值为x的节点
    if(o == null) {
        o = &nodes[cnt++];
        o->ch[0] = o->ch[1] = null;
        o->v = x; o->r = rand(); o->s = 1;
    } else {
        int d = o->cmp(x);
        ins(o->ch[d], x); if(o->ch[d]->r > o->r) rot(o, d^1);
    }
    o->maintain();
}

void del(node *&o, int x) { //删掉值为x的节点，调用前保证节点存在
    int d = o->cmp(x);
    if(d == -1) {
        if(o->ch[0] == null) o = o->ch[1];
        else if(o->ch[1] == null) o = o->ch[0];
        else {
            int d2 = o->ch[0]->r > o->ch[1]->r? 1: 0;
            rot(o, d2); del(o->ch[d2], x);
        }
    } else {
        del(o->ch[d], x);
    }
    if(o != null) o->maintain();
}

int mcount(node *p, int v) { //返回以p为根的名次树中大于v的个数
    if(p == null) return 0;
    if (p->v > v) return 1+p->ch[1]->s+mcount (p->ch[0], v);
    else return mcount (p->ch[1], v);
}

int n, m, a[maxn];

void init () {
    srand ((long long)time (0));
    null = &nodes[0]; null->s = 0;
    cnt = 1;
    for (int i = 1; i <= n; i++) {
        root[i] = null;
        a[i] = i;
    }
    for (int i = 1; i <= n; i++) {
        for (int j = i; j <= n; j += lowbit (j)) ins (root[j], a[i]);
    }
}

long long sum = 0;

int query (int x, int val) {//查询x子树中大于val的个数
    int ans = 0;
    for (int i = x; i; i -= lowbit (i)) {
        ans += mcount (root[i], val);
    }   return ans;
}

int main () {
    cin >> n >> m;
    init ();
    for (int i = 0; i < m; i++) {
        int l, r; scan (l); scan (r);
        if (r < l) swap (l, r);
        if (l == r) {
            printf ("%lld\n", sum);
            continue;
        }
        int tmp1 = query (l, a[l]);
        int tmp2 = query (r, a[r]);
        if (a[r] < a[l]) tmp2--;
        for (int i = l; i <= n; i += lowbit (i)) del (root[i], a[l]);
        for (int i = r; i <= n; i += lowbit (i)) del (root[i], a[r]);
        for (int i = l; i <= n; i += lowbit (i)) ins (root[i], a[r]);
        for (int i = r; i <= n; i += lowbit (i)) ins (root[i], a[l]);
        int tmp3 = query (l, a[r]);
        int tmp4 = query (r, a[l]);
        if (a[r] > a[l]) tmp4--;
        sum += ((tmp4-tmp1)-(r-l-1-(tmp4-tmp1))+r-l-1-2*(tmp2-tmp3));
        if (a[r] < a[l]) sum--;
        else sum++;
        swap (a[l], a[r]);
        printf ("%lld\n", sum);
    }
    return 0;
}
```

splay树

`BZOJ 1552`:每次翻转区间[i,pos]，pos表示除了前i-1个数外最小的数的位置，输出每次翻转的pos。

```
struct node;
node *null; //安全空指针
struct node {
    node *ch[2], *fa;
    int sz, rev; //子树size  区间翻转标记
    node () {
        ch[0] = ch[1] = null; rev = 0;
    }
    inline void push_up () {
        if (this == null) return ;
        sz = ch[0]->sz+ch[1]->sz+1;
    }
    inline void setc (node *p, int d) { //设置p是这个节点的d孩子
        ch[d] = p;
        p->fa = this;
    }
    inline bool d () { //判断是否是右孩子 0:左孩子 1:右孩子
        return fa->ch[1] == this;
    }
    void clear () { //清空这个节点
        sz = 1;
        ch[0] = ch[1] = fa = null;
        rev = 0;
    }
    void update_rev () { //这个节点进行左右逆转
        if (this == null) return ;
        swap (ch[0], ch[1]);
        rev ^= 1;
    }
    inline void push_down () { //lazy标记下放
        if (this == null) return ;
        if (rev) {
            ch[0]->update_rev ();
            ch[1]->update_rev ();
            rev = 0;
        }
    }
    inline bool is_root () { //判断是否是根
        return fa == null || (this != fa->ch[0] && this != fa->ch[1]);
    }
}pool[maxn], *tail, *no[maxn], *root;

inline void rotate (node *x) { //将节点x进行一次双旋
    node *f = x->fa, *ff = x->fa->fa;
    f->push_down (); x->push_down ();
    int c = x->d(), cc = f->d ();
    f->setc (x->ch[!c], c);
    x->setc (f, !c);
    if (ff->ch[cc] == f) ff->setc (x, cc);
    else x->fa = ff;
    f->push_up ();
}

inline void splay (node *&root, node *x, node *goal) {//将x旋转为goal的孩子 goal为null表
示旋转为根
    while (x->fa != goal) {
        if (x->fa->fa == goal) rotate (x);
        else {
            x->fa->fa->push_down ();
            x->fa->push_down ();
            x->push_down ();
            bool f = x->fa->d();
            x->d () == f ? rotate (x->fa) : rotate (x);
            rotate (x);
        }
    }
    x->push_up ();
    if (goal == null) root = x;
}
```

```
node *get_kth (node *r, int k) {//找到从r出发的第k个元素
    node *x = r;
    x->push_down ();
    while (x->ch[0]->sz+1 != k) {
        if (k < x->ch[0]->sz+1) x = x->ch[0];
        else {
            k -= x->ch[0]->sz+1;
            x = x->ch[1];
        }
        x->push_down ();
    }
    return x;
}

node *get_next (node *p) { //找到p的后继
    p->push_down ();
    p = p->ch[1];
    p->push_down ();
    while (p->ch[0] != null) {
        p = p->ch[0];
        p->push_down ();
    }
    return p;
}

node *get_pre (node *p) { //找p的前驱
    p->push_down ();
    p = p->ch[0];
    p->push_down ();
    while (p->ch[1] != null) {
        p = p->ch[1];
        p->push_down ();
    }
    return p;
}

void build_tree (node *&x, int l, int r, node *fa) {
    if (l > r) return ;
    int mid = (l+r)>>1;
    x = tail++;
    x->clear (); x->fa = fa;
    no[mid] = x;
    build_tree (x->ch[0], l, mid-1, x);
    build_tree (x->ch[1], mid+1, r, x);
    x->push_up ();
}

void init (int n) { //给一个n个节点的splay初始化
    tail = pool;
    null = tail++; null->fa = null->ch[0] = null->ch[1] = null;
    null->sz = 0; null->rev = 0;
    node *p = tail++;
    p->clear (); root = p;
    p = tail++; p->clear (); root->setc (p, 1);
    build_tree (root->ch[1]->ch[0], 1, n, root->ch[1]);
    root->ch[1]->push_up ();
    root->push_up ();
}
int a[maxn], b[maxn], n;
int main () {
    while (cin >> n && n) {
        for (int i = 1; i <= n; i++) {
            scanf ("%d", &a[i]);
            b[i] = i;
        }
        sort (b+1, b+n+1, [&](int i, int j) {return a[i] < a[j] || (a[i] == a[j] && i
< j);});
        init (n);
        for (int i = 1; i <= n; i++) {
            splay (root, no[b[i]], null);
            int sz = root->ch[0]->sz;
            printf ("%d%c", sz, i == n ? '\n' : ' ');
            splay (root, get_kth (root, i), null);
            splay (root, get_kth (root, sz+2), root);
            root->ch[1]->ch[0]->update_rev();
        }
    }
    return 0;
}
```

树分治

一类树路径问题

大部分用于解决树上路径问题。当每次从重心处理整个树时能保证分治树高在logn范围中，可以在定下重心时暴力遍历子树。

`HDU 4812`:求一个树上路径乘积模m为k的最小字典序点对。

```
int n, k;
struct node {
    int u, v, next;
    node (int _u = 0, int _v = 0, int _next = 0) {
        u = _u, v= _v, next = _next;
    }
}edge[maxm];
int head[maxn], cnt, w[maxn];

void add_edge (int u, int v) {
    edge[cnt] = node (u, v, head[u]); head[u] = cnt++;
```

```
}

pii ans;
bool vis[maxn]; int sz[maxn], son[maxn];
long long rev[mod+5];
long long qpow (long long a, long long b, long long p) {
    if (b == 0) return 1;
    long long ans = qpow (a, b>>1, p);
    ans = ans*ans%p;
    if (b&1) ans = ans*a%p;
    return ans;
}

void get_sz (int u, int fa) { //处理子树size
    sz[u] = 1, son[u] = 0;
    for (int i = head[u]; i+1; i = edge[i].next) {
        int v = edge[i].v; if (v == fa || vis[v]) continue;
        get_sz (v, u);
        sz[u] += sz[v];
        get_max (son[u], sz[v]);
    }
}

void find_root (int u, int fa, int &root, int &tmp, int &tot) { //找到重心
    if (max (son[u], tot-sz[u]) < tmp) {
        tmp = max (son[u], tot-sz[u]);
        root = u;
    }
    for (int i = head[u]; i+1; i = edge[i].next) {
        int v = edge[i].v; if (vis[v] || v == fa) continue;
        find_root (v, u, root, tmp, tot);
    }
}

pii tmp[maxn]; int id;
int mark[mod+5]; //每个余数对应的链的另一端

void get_dis (int u, int fa, int d) { //这个重心下孩子到他的距离
    d = 1LL*d*w[u]%mod;
    tmp[id++] = make_pair (d, u);
    for (int i = head[u]; i+1; i =edge[i].next) {
        int v = edge[i].v; if (v == fa || vis[v]) continue;
        get_dis (v, u, d);
    }
}

void clean (int u, int fa, int d) { //消除这个重心下的孩子的影响
    d = 1LL*d*w[u]%mod;
    mark[d] = INF;
    for (int i = head[u]; i+1; i = edge[i].next) {
        int v = edge[i].v; if (v == fa || vis[v]) continue;
```

```
            clean (v, u, d);
        }
}

void solve (int u) {
        get_sz (u, 0);
        int root, cur = INF;
        find_root (u, 0, root, cur, sz[u]);
        vis[root] = 1; mark[w[root]] = root;
        for (int i = head[root]; i+1; i =edge[i].next) {
            int v = edge[i].v; if (vis[v]) continue;
            id = 0;
            get_dis (v, root, 1);
            for (int j = 0; j < id; j++) if (tmp[j].fi != 0) {
                int need = 1LL*k*rev[tmp[j].fi]%mod;
                if (mark[need] != INF) {
                    get_min (ans, make_pair (min (tmp[j].se, mark[need]), max (tmp[j].se, 
mark[need])));
                }
            }
            for (int j = 0; j < id; j++) {
                get_min (mark[1LL*tmp[j].fi*w[root]%mod], tmp[j].se);
            }
        }
        clean (root, 0, 1);
        for (int i = head[root]; i+1; i = edge[i].next) {
            int v = edge[i].v; if (vis[v]) continue;
            solve (v);
        }
}

int main () {
        for (int i = 1; i < mod; i++) rev[i] = qpow (i, mod-2, mod);
        for (int i = 0; i < mod; i++) mark[i] = INF;
        while (scanf("%d%d", &n, &k) == 2) {
            memset (head, -1, (n+5)*sizeof head[0]); cnt = 0; ans = make_pair (INF, INF);
            for (int i = 1; i <= n; i++) scanf ("%d", &w[i]);
            for (int i = 1; i < n; i++) {
                int u, v; scanf ("%d%d", &u, &v);
                add_edge (u,v ); add_edge (v, u);
            }
            memset (vis, 0, (n+5)*sizeof vis[0]);
            solve (1);
            if (ans == make_pair (INF, INF)) printf ("No solution\n");
            else printf ("%d %d\n", ans.fi, ans.se);
        }
        return 0;
}
```

点分治序列

虚树

虚树题一般的类型：有q个询问每次询问k个关键点，保证$\sum_{i=0}^{q} k_i \leq n$，并且每次询问都能通过一次遍历整棵树的dp求出。 可以通过把关键点（询问的点和所有两两询问的LCA）创建成一颗虚树，通过一次虚树上的DP完成对答案的计算。

`BZOJ 2286`:给出一个无向带权图，每次询问k个点，问从0不能到达任何其中的点需要切断的边权的最小值。

用一个栈构建虚树，出栈的时候更新DP即可。求所有关键点可以通过dfs序排序，然后两两相邻求LCA即可。

```
const int DEG = 20;//大约是lgn
struct Edge {
    int to, next, w;
}edge[maxm];
int n, m; long long dp[maxn], to[maxn];
bool is[maxn], vis[maxn];
int head[maxn], tot;
void add_edge(int u,int v, int w) {
    edge[tot].to = v;
    edge[tot].w = w;
    edge[tot].next = head[u];
    head[u] = tot++;
}
void init () {
    tot = 0;
    memset (head, -1, sizeof head);
}
int fa[maxn][DEG];//fa[i][j]表示结点i的第2^j个祖先
int deg[maxn];//深度数组
int LCA (int u,int v) {
    if (deg[u] > deg[v]) swap (u, v);
    int hu = deg[u], hv = deg[v];
    int tu = u, tv = v;
    for(int det = hv-hu, i = 0; det; det >>= 1, i++) {
        if (det&1)
            tv = fa[tv][i];
        if (tu == tv)
            return tu;
    }
    for(int i = DEG-1; i >= 0; i--) {
        if (fa[tu][i] == fa[tv][i])
            continue;
        tu = fa[tu][i];
        tv = fa[tv][i];
    }
    return fa[tu][0];
}

int dfs_clock, dfn[maxn];
int Top[maxn], num[maxn], p[maxn], son[maxn];//重链的顶端 子树的size v和父亲的边对应的编号 重儿子
int pos;
void dfs1 (int u, int d, int father) { //当前的点 当前的深度
    dfn[u] = ++dfs_clock;
    num[u] = 1;
    deg[u] = d;
    for (int i = 1; i < DEG; i++) {
        fa[u][i] = fa[fa[u][i-1]][i-1];
    }
    for (int i = head[u]; i+1; i = edge[i].next) {
        int v = edge[i].to; if (v == father) continue;
        fa[v][0] = u;
        to[v] = edge[i].w;
        dfs1 (v, d+1, u);
        num[u] += num[v];
        if (son[u] == -1 || num[v] > num[son[u]]) {
            son[u] = v;
        }
    }
}

void dfs2 (int u, int sp) {
    Top[u] = sp;
    p[u] = ++pos;
    if (son[u] == -1) return ;
    dfs2 (son[u], sp);
    for (int i = head[u]; i+1; i = edge[i].next) {
        int v = edge[i].to; if (v == fa[u][0] || v == son[u]) continue;
        dfs2 (v, v);
    }
}

long long g[maxn][DEG];
void rmq_init () {
    to[1] = 1e15;
    for (int i = 1; i <= n; i++) {
        g[p[i]][0] = to[i];
    }
    for (int j = 1; (1<<j) <= n; j++) {
        for (int i = 0; i+(1<<j)-1 <= n; i++) {
            g[i][j] = min (g[i][j-1], g[i+(1<<(j-1))][j-1]);
        }
    }
}

long long rmq (int l, int r) {
    if (l > r) swap (l, r);
    int k = 0;
    while ((1<<(k+1)) <= r-l+1) k++;
    return min (g[l][k], g[r-(1<<k)+1][k]);
}
```

```
long long f (int u, int v) { //查询u到v的边的最小值
    if (u == v) return 0;
    int f1 = Top[u], f2 = Top[v];
    long long tmp = 1e15;
    while (f1 != f2) {
        if (deg[f1] < deg[f2]) {
            swap (f1, f2);
            swap (u, v);
        }
        tmp = min (tmp, rmq (p[f1], p[u]));
        u = fa[f1][0], f1 = Top[u];
    }
    if (u == v) return tmp;
    if (deg[u] > deg[v]) swap (u, v);
    return min (tmp, rmq (p[son[u]], p[v]));
}


int a[maxn]; int id, k;
bool cmp (const int &a, const int &b) { //按照dfs序排序
    return dfn[a] < dfn[b];
}

int s[maxn], top;
void solve () {
    top = 0; dp[1] = 0;
    for (int i = 0; i < k; i++) {
        if (top == 0 || top == 1) {
            s[++top] = a[i];
            continue;
        }
        int lca;
        while (lca = LCA (s[top], a[i]), lca != s[top]) {
            int u = s[top], p = s[top-1];
            if (is[u]) {
                dp[u] = to[u];
                dp[p] = min (dp[p]+f (u, p), to[p]);
            }
            else {
                dp[p] = min (min (dp[p]+dp[u], dp[p]+f (u, p)),  to[p]);
            }
            top--;
        }
        s[++top] = a[i];
    }
    while (top > 1) {
        int u = s[top], p = s[top-1];
        if (is[u]) {
            dp[u] = to[u];
            dp[p] = min (dp[p]+f (u, p), to[p]);
        }
        else {
            dp[p] = min (min (dp[p]+dp[u], dp[p]+f (u, p)),  to[p]);
        }
        top--;
    }
    printf ("%lld\n", dp[1]);

    for (int i = 0; i < k; i++) {
        is[a[i]] = 0;
        dp[a[i]] = 0;
        vis[a[i]] = 0;
    }
}

int main () {
    //Open ();
    while (scanf ("%d", &n) == 1) {
        init ();
        for (int i = 1; i < n; i++) {
            int u, v, w;
            scan (u); scan (v); scan (w);
            add_edge (u, v, w);
            add_edge (v, u, w);
        }
        dfs_clock = 0;
        pos = 0; Clear (son, -1);
        fa[1][0] = 1;
        to[1] = 1e15;
        dfs1 (1, 0, 0);
        dfs2 (1, 1);
        rmq_init ();

        Clear (is, 0); Clear (dp, 0); Clear (vis, 0);
        scan (m);
        while (m--) {
            scan (k); id = k;
            for (int i = 0; i < k; i++) {
                int tmp; scan (tmp);
                a[i] = tmp; is[tmp] = 1; vis[tmp] = 1;
            }
            sort (a, a+k, cmp);
            for (int i = 0; i < k-1; i++) {
                int lca = LCA (a[i], a[i+1]);
                if (!vis[lca]) {
                    a[id++] = lca;
                    vis[lca] = 1;
                }
            }
            if (!vis[1]) a[id++] = 1;
            sort (a, a+id, cmp);
            k = id;
```

```
                solve ();
            }
        }
    return 0;
}
```

左偏树

支持O(nlgn)合并的优先队列。

注意:

- 用delete释放内存;
- 如果不需要释放内存仅仅只是清空指针直接赋值NULL。

```
struct node {//左偏树节点
    int key, dis;
    node *l, *r;
    bool operator < (const node &b) const {
        //重载<运算符
        return key > b.key;
    }
};

void init (node *&t, int key) {
    t = new node;
    t->l = t->r = NULL;
    t->key = key;
    t->dis = 0;
}

int dis (node *a) {//计算左偏树的dis值
    if (a == NULL)
        return -1;
    return a->dis;
}

node *merge (node *a, node *b) {//合并两棵左偏树
    //返回新的根节点指针
    if (a == NULL) return b;
    if (b == NULL) return a;
    if (*b < *a) swap (a, b);
    a->r = merge (a->r, b);
    if (dis (a->r) > dis (a->l)) swap (a->l, a->r);
    a->dis = dis (a->r) + 1;
    return a;
}

node *pop (node *a) {//删除左偏树key值最小的节点
    //返回新的根节点指针
    if (!a) return NULL;
    return merge (a->l, a->r);
}

node *insert (node *root, int num) {//左偏树中插入key值为num的节点
    //返回新的根节点指针
    node *p;
    init (p, num);
    return merge (root, p);
}

int top (node *a) {//去取左偏树key值最大的元素
    //返回值
    return a->key;
}
```

`HDU 3512`：给定一个序列，给每一个数+1或者-1，求最少的操作次数使得序列非递减。 拓展：使得序列严格递减，只需要给每一个数预先减去他的下标即可。

```
#define Maxn 2500
using namespace std;
const int inf = 0x7fffffff;
struct Tnode
{
       int key, dist;
       Tnode *lc, *rc;
       Tnode (int x)
       :key(x),dist(0),lc(NULL),rc(NULL){}
       };
class leftist
{
private:
        int tot;
        Tnode *root;
        inline int Get_dis(Tnode *x) {return x==NULL?-1:x->dist;}
        inline Tnode* Merge(Tnode *a, Tnode *b)
        {
            if (!a) return b;
            if (!b) return a;
            if (a->key < b->key) swap(a, b);
            a->rc = Merge(a->rc, b);
            if (Get_dis(a->rc) > Get_dis(a->lc)) swap(a->lc, a->rc);
            a->dist = Get_dis(a->rc) + 1;
            return a;
        }
public:
        inline int Size() {return tot;}
        inline bool empty() {return tot==0;}
        inline void clr() {root=NULL;tot=0;}
        inline Tnode* groot(){return root;}
        inline void insert(int x){root = Merge(root, new Tnode(x));tot++;}
        inline int top() {return root!=NULL?root->key:-inf;}
        inline void pop()
        {
            Tnode *tmp = root;
            root = Merge(root->lc, root->rc);
            delete tmp; tot--;
        }
        inline void merge(leftist &t)
        {
            tot += t.Size();
            root = Merge(root, t.groot());
            t.clr();
        }
      };
int n, d[Maxn], len[Maxn];
leftist stk[Maxn];
int main()
{
    int hi=0;
    scanf("%d", &n);
    for (int i=0; i<=n; i++) stk[i].clr();
    for (int i=0; i<n; i++)
    {
        scanf("%d", &d[i]);
        leftist now;
        now.clr(); now.insert(d[i]);
        int ct = 1;
        while (hi>0 && stk[hi-1].top() > now.top())
        {
            now.merge(stk[--hi]);
            if ((len[hi]+1)/2 + (ct+1)/2 > (len[hi]+ct+1) / 2) now.pop();
            ct += len[hi];
        }
        len[hi] = ct;
        stk[hi++] = now;
    }
    long long ans = 0;
    int idx = 0;
    for (int i=0; i<hi; i++)
    {
      int num = stk[i].top();
      for (;len[i]; len[i]--, idx++)
        ans += abs(d[idx] - num);
    }
    cout <<ans <<endl;
    return 0;
}
```

KD树

`BZOJ2716`:给出n个点，接下来m个操作，每次插入一个点，或者询问离询问点的最近曼哈顿距离。

暴力插点询问

```
struct node {
    int d[2], l, r;//节点的点的坐标 左右孩子
    int Max[2], Min[2];//节点中点x的最值 y的最值
}tree[maxn<<1], tmp;
int n, m;
int root, cmp_d;

bool cmp (const node &a, const node &b) {
    return a.d[cmp_d] < b.d[cmp_d] || (a.d[cmp_d] == b.d[cmp_d] &&
            a.d[cmp_d^1] < b.d[cmp_d^1]);
}

void push_up (int p, int pp) {
    get_min (tree[p].Min[0], tree[pp].Min[0]);
    get_min (tree[p].Min[1], tree[pp].Min[1]);
    get_max (tree[p].Max[0], tree[pp].Max[0]);
    get_max (tree[p].Max[1], tree[pp].Max[1]);
}

int build_tree (int l, int r, int D) {
    int mid = (l+r)>>1;
    tree[mid].l = tree[mid].r = 0;
    cmp_d = D;
    nth_element (tree+l+1, tree+mid+1, tree+1+r, cmp);
    //按照cmp把第mid元素放在中间 比他小的放左边 比他大的放右边
    tree[mid].Max[0] = tree[mid].Min[0] = tree[mid].d[0];
    tree[mid].Max[1] = tree[mid].Min[1] = tree[mid].d[1];
    if (l != mid) tree[mid].l = build_tree (l, mid-1, D^1);
    if (r != mid) tree[mid].r = build_tree (mid+1, r, D^1);
    if (tree[mid].l) push_up (mid, tree[mid].l);
    if (tree[mid].r) push_up (mid, tree[mid].r);
    return mid;
}

void insert (int now) {
    int D = 0, p = root;
    while (1) {
        push_up (p, now);//先更新p节点
        if (tree[now].d[D] >= tree[p].d[D]) {
            if (!tree[p].r) {
                tree[p].r = now;
                return ;
            }
            else p = tree[p].r;
        }
        else {
            if (!tree[p].l) {
                tree[p].l = now;
                return ;
            }
            else p = tree[p].l;
        }
        D ^= 1;
    }
    return ;
}

#define INF 1e9
int ans, x, y;
int dis (int p, int x, int y) {//点(x,y)在p的管辖范围内的可能最小值
    int ans = 0;
    if (x < tree[p].Min[0]) ans += tree[p].Min[0]-x;
    if (x > tree[p].Max[0]) ans += x-tree[p].Max[0];
    if (y < tree[p].Min[1]) ans += tree[p].Min[1]-y;
    if (y > tree[p].Max[1]) ans += y-tree[p].Max[1];
    return ans;
}

void query (int p) {
    int dl = INF, dr = INF, d0;
    d0 = abs (tree[p].d[0]-x) + abs (tree[p].d[1]-y);//初始答案
    get_min (ans, d0);
    if (tree[p].l) dl = dis (tree[p].l, x, y);
    if (tree[p].r) dr = dis (tree[p].r, x, y);
    if (dl < dr) {
        if (dl < ans) query (tree[p].l);
        if (dr < ans) query (tree[p].r);
    }
    else {
        if (dr < ans) query (tree[p].r);
        if (dl < ans) query (tree[p].l);
    }
}

int main () {
    //Open ();
    while (scanf ("%d%d", &n, &m) == 2) {
        for (int i = 1; i <= n; i++) {
            scan (tree[i].d[0]);
            scan (tree[i].d[1]);
        }
        root = build_tree (1, n, 0);
        for (int i = 1; i <= m; i++) {
            int op;
            scan (op); scan (x); scan (y);
            if (op == 1) {
                n++; tree[n].d[0] = x, tree[n].d[1] = y;
                tree[n].Max[0] = tree[n].Min[0] = x;
                tree[n].Max[1] = tree[n].Min[1] = y;
                insert (n);
            }
            else {
                ans = INF;
                query (root);
                printf ("%d\n", ans);
            }
        }
    }
    return 0;
}
```

`codeforces(?)`：k维坐标系下的最近点对问题。直接对于每一个询问都在kdtree中询问m次最近点，每次找到一个最近点对需要把它记录下来，当下次再找到它的时候距离直接设置成无穷大即可。

```
struct node {
    long long d[5];
    int l, r;//节点的点的坐标  左右孩子
    long long Max[5], Min[5];//节点中点x的最值  y的最值
    int id;
}tree[maxn<<1], tmp;
int n, m, k;
int root, cmp_d;

bool cmp (const node &a, const node &b) {
    return a.d[cmp_d] < b.d[cmp_d] || (a.d[cmp_d] == b.d[cmp_d] &&
            a.d[cmp_d^1] < b.d[cmp_d^1]);
}

void push_up (int p, int pp) {
    for (int i = 0; i < k; i++) {
        get_min (tree[p].Min[i], tree[pp].Min[i]);
        get_max (tree[p].Max[i], tree[pp].Max[i]);
    }
}

int build_tree (int l, int r, int D) {
    int mid = (l+r)>>1;
    tree[mid].l = tree[mid].r = 0;
    cmp_d = D;
    nth_element (tree+l+1, tree+mid+1, tree+1+r, cmp);
    //按照cmp把第mid元素放在中间 比他小的放左边 比他大的放右边
    for (int i = 0; i < k; i++) {
        tree[mid].Max[i] = tree[mid].Min[i] = tree[mid].d[i];
    }
    if (l != mid) tree[mid].l = build_tree (l, mid-1, (D+1)%k);
    if (r != mid) tree[mid].r = build_tree (mid+1, r, (D+1)%k);
    if (tree[mid].l) push_up (mid, tree[mid].l);
    if (tree[mid].r) push_up (mid, tree[mid].r);
    return mid;
}

void insert (int now) {
    int D = 0, p = root;
    while (1) {
        push_up (p, now);//先更新p节点
        if (tree[now].d[D] >= tree[p].d[D]) {
            if (!tree[p].r) {
                tree[p].r = now;
                return ;
            }
            else p = tree[p].r;
        }
        else {
            if (!tree[p].l) {
                tree[p].l = now;
                return ;
            }
            else p = tree[p].l;
        }
        D = (D+1)%k;
    }
    return ;
}

#define INF 4e18
#define sqr(a) (a)*(a)
pli ans;
long long x[5];
bool vis[maxn];
long long dis (int p) {//点在p的管辖范围内的可能最小值
    long long ans = 0;
    for (int i = 0; i < k; i++) {
        if (x[i] < tree[p].Min[i]) ans += sqr (tree[p].Min[i]-x[i]);
        if (x[i] > tree[p].Max[i]) ans += sqr (x[i]-tree[p].Max[i]);
    }
    return ans;
}

long long distance (int p) {
    long long ans = 0;
    for (int i = 0; i < k; i++) {
        double tmp = tree[p].d[i]-x[i];
        ans += tmp*tmp;
    }
    return ans;
}

void query (int p) {
    long long dl = INF, dr = INF, d0 = INF;
    if (!vis[tree[p].id])
        d0 = distance (p);//初始答案 排除重合
    if (d0 < ans.fi) ans = make_pair (d0, tree[p].id);
    if (tree[p].l) dl = dis (tree[p].l);
    if (tree[p].r) dr = dis (tree[p].r);
    if (dl < dr) {
        if (dl < ans.fi) query (tree[p].l);
        if (dr < ans.fi) query (tree[p].r);
    }
    else {
        if (dr < ans.fi) query (tree[p].r);
        if (dl < ans.fi) query (tree[p].l);
    }
}

long long res[maxn];
struct Point {
    long long x[5], id;
    bool operator < (const Point a) const {
        for (int i = 0; i < k; i++) if (x[i] != a.x[i]) {
            return x[i] < a.x[i];
        }
        return id < a.id;
    }
}point[maxn], del[15];

int main () {
    //Open ();
    while (scanf ("%d%d", &n, &k) == 2) {
        for (int i = 1; i <= n; i++) {
            for (int j = 0; j < k; j++) {
                scan (tree[i].d[j]);
                point[i].x[j] = tree[i].d[j];
            }
            tree[i].id = point[i].id = i;
        }
        root = build_tree (1, n, 0);
        scanf ("%d", &m);
        Clear (vis, 0);
        for (int i = 1; i <= m; i++) {
            for (int j = 0; j < k; j++) scan (x[j]);
            int cnt; scan (cnt);
            for (int j = 0; j < cnt; j++) {
                ans = make_pair (INF, 0);
                query (root);
                del[j] = point[ans.se];
                vis[ans.se] = 1;
            }
            printf ("the closest %d points are:\n", cnt);
            for (int j = 0; j < cnt; j++) {
                vis[del[j].id] = 0;
                for (int l = 0; l < k; l++) {
                    printf ("%d", del[j].x[l]);
                    if (l < k-1) printf (" ");
                }
                printf ("\n");
            }
        }
    }
    return 0;
}
```

`HDU 5992`:给出n个酒店，每个酒店有一个花费和坐标。然后给出m个询问，输出离询问最近并且花费在询问要求内的酒店。

第一个想法是两种东西按照花费排序，每次插入新酒店。但是这个插入比较麻烦，在kdtree退化的时候需要及时重构（套个替罪羊树）;还有一种就是直接建三维kdtree，然后对于每一个询问，如果一个节点范围内最小第三维比询问大，那么可以直接忽略。计算的时候只要算上两维的距离即可。

```
struct node {
    long long d[3];
    int l, r, id;//节点的点的坐标 左右孩子
    long long Max[3], Min[3];//节点中点x的最值 y的最值
}tree[maxn<<1], tmp;
int n, m;
int root, cmp_d;
struct Point {
    long long x, y, c;
    int id;
}p1[maxn], p2[maxn], pre[maxn];

bool cmp (const node &a, const node &b) {
    for (int i = 0; i < 3; i++) if (a.d[i] != b.d[i]) return a.d[i] < b.d[i];
    return a.id < b.id;
}

void push_up (int p, int pp) {
    for (int i = 0; i < 3; i++) {
        get_min (tree[p].Min[i], tree[pp].Min[i]);
        get_max (tree[p].Max[i], tree[pp].Max[i]);
    }
}

int build_tree (int l, int r, int D) {
    int mid = (l+r)>>1;
    tree[mid].l = tree[mid].r = 0;
    cmp_d = D;
    nth_element (tree+l+1, tree+mid+1, tree+1+r, cmp);
    //按照cmp把第mid元素放在中间 比他小的放左边 比他大的放右边
    for (int i = 0; i < 3; i++) {
        tree[mid].Max[i] = tree[mid].Min[i] = tree[mid].d[i];
    }
    if (l != mid) tree[mid].l = build_tree (l, mid-1, (D+1)%3);
    if (r != mid) tree[mid].r = build_tree (mid+1, r, (D+1)%3);
    if (tree[mid].l) push_up (mid, tree[mid].l);
    if (tree[mid].r) push_up (mid, tree[mid].r);
    return mid;
}

void insert (int now) {
    int D = 0, p = root;
    while (1) {
        push_up (p, now);//先更新p节点
        if (tree[now].d[D] >= tree[p].d[D]) {
            if (!tree[p].r) {
                tree[p].r = now;
                return ;
            }
            else p = tree[p].r;
        }
        else {
            if (!tree[p].l) {
                tree[p].l = now;
                return ;
            }
            else p = tree[p].l;
        }
        D ^= 1;
    }
    return ;
}

#define INF 0x3f3f3f3f3f3f
#define sqr(a) (a)*(a)
pli ans;
long long x, y, z;
long long dis (int p) {//点(x,y)在p的管辖范围内的可能最小值
    long long ans = 0;
    if (z < tree[p].Min[2]) return INF;
    //if (z > tree[p].Max[2]) return sqr (z-tree[p].Max[2]);
    if (x < tree[p].Min[0]) ans += sqr (tree[p].Min[0]-x);
    if (x > tree[p].Max[0]) ans += sqr (x-tree[p].Max[0]);
    if (y < tree[p].Min[1]) ans += sqr (tree[p].Min[1]-y);
    if (y > tree[p].Max[1]) ans += sqr (y-tree[p].Max[1]);
    return ans;
}

void query (int p) {
    pli dl = make_pair (INF, 1), dr = make_pair (INF, 1);
    long long d0 = INF;
    if (z >= tree[p].d[2])
```

```
        d0 = sqr (tree[p].d[0]-x) + sqr (tree[p].d[1]-y);//初始答案
    get_min (ans, make_pair (d0, tree[p].id));
    if (tree[p].l) dl = make_pair (dis (tree[p].l), tree[tree[p].l].id);
    if (tree[p].r) dr = make_pair (dis (tree[p].r), tree[tree[p].r].id);
    if (dl < dr) {
        if (dl < ans) query (tree[p].l);
        if (dr < ans) query (tree[p].r);
    }
    else {
        if (dr < ans) query (tree[p].r);
        if (dl < ans) query (tree[p].l);
    }
}

int main () {
    int t; scan (t);
    while (t--) {
        scan (n), scan (m);
        for (int i = 1; i <= n; i++) {
            scan (pre[i].x); scan (pre[i].y); scan (pre[i].c); pre[i].id = i;
            p1[i] = pre[i];
            tree[i].d[0] = pre[i].x, tree[i].d[1] = pre[i].y, tree[i].d[2] =
pre[i].c;
            tree[i].id = i;
        }
        for (int i = 1; i <= m; i++) {
            scan (p2[i].x); scan (p2[i].y); scan (p2[i].c);
        }
        root = build_tree (1, n, 0);
        for (int i = 1; i <= m; i++) {
            ans = make_pair (INF, 0); x = p2[i].x, y = p2[i].y, z = p2[i].c;
            query (root); int id = ans.se;
            printf ("%lld %lld %lld\n", pre[id].x, pre[id].y, pre[id].c);
        }
    }
    return 0;
}
```

莫队

如果查询离线没有修改，并且能够O(1)时间由[l,r]的结果知道[l-1,r],[l+1,r],[l,r-1],[l,r+1]的结果，就可以用莫队$O(n\sqrt{n})$。

`BZOJ 2038`：给出一串数字，每次询问区间中任取两只袜子能够配对的概率。

```
#define maxn 51111

int pos[maxn];
int n, m, k;
long long cnt[maxn];
long long ans[maxn], tmp[maxn];
struct node {
    int l, r, id;
    bool operator < (const node &a) const {
        return pos[l] < pos[a.l] || (pos[l] == pos[a.l] && r < a.r);
    }
} p[maxn];
int a[maxn];
long long cur;

void add (int pos) {
    cur += cnt[a[pos]];
    cnt[a[pos]]++;
}

void del (int pos) {
    cnt[a[pos]]--;
    cur -= cnt[a[pos]];
}

int main () {
    //freopen ("in.txt", "r", stdin);
    while (scanf ("%d%d", &n, &m) == 2) {
        a[0] = 0;
        for (int i = 1; i <= n; i++) {
            scanf ("%d", &a[i]);
        }
        int block = ceil (sqrt (n*1.0));
        for (int i = 1; i <= n; i++) pos[i] = (i-1)/block;
        for (int i = 0; i < m; i++) {
            scanf ("%d%d", &p[i].l, &p[i].r);
            p[i].id = i;
        }
        sort (p, p+m);
        int l = 1, r = 1;
        cur = 0;
        memset (cnt, 0, sizeof cnt);
        add (1);
        for (int i = 0; i < m; i++) {
            while (r > p[i].r) {
                del (r);
                r--;
            }
            while (r < p[i].r) {
                r++;
                add (r);
            }
            while (l > p[i].l) {
                l--;
                add (l);
            }
            while (l < p[i].l) {
                del (l);
                l++;
            }
            ans[p[i].id] = cur;
            tmp[p[i].id] = p[i].r-p[i].l+1;
        }
        for (int i = 0; i < m; i++) {
            tmp[i] = tmp[i]*(tmp[i]-1)/2;
            if (ans[i] == 0) {
                printf ("0/1\n");
                continue;
            }
            long long g = __gcd (tmp[i], ans[i]);
            printf ("%lld/%lld\n", ans[i]/g, tmp[i]/g);
        }
    }
    return 0;
}
```

主席树

`HDU 2556`：静态区间第k小

```
#define maxn 111111

struct node {
    int l, r;
    int num;
}tree[maxn<<6];
int T[maxn];
int cnt;//总结点数
int n, q, a[maxn];
vector <int> num;
int p[maxn];
map <int , int> gg;

void init () {
    sort (num.begin (), num.end ());
    int cnt = 0;
    for (int i = 0; i < num.size (); i++) {
        if (!i || num[i] != num[i-1]) {
            gg[num[i]] = ++cnt;
            p[cnt] = num[i];
        }
    }
}

int build_tree (int l, int r) {
    int root = cnt++;
    tree[root].num = 0;
    if (l == r)
        return root;
    int mid = (l+r)>>1;
    tree[root].l = build_tree (l, mid);
    tree[root].r = build_tree (mid+1, r);
    return root;
}

int update (int root, int pos, int val) {
    int new_root = cnt++;
    int tmp = new_root;
    tree[new_root].num = tree[root].num+val;
    int l = 1, r = n;
    while (l < r) {
        int mid = (l+r)>>1;
        if (pos <= mid) {
            tree[new_root].l = cnt++;
            tree[new_root].r = tree[root].r;
            new_root = tree[new_root].l;
            root = tree[root].l;
            r = mid;
        }
        else {
            tree[new_root].l = tree[root].l;
            tree[new_root].r = cnt++;
            new_root = tree[new_root].r;
            root = tree[root].r;
            l = mid+1;
        }
        tree[new_root].num = tree[root].num + val;
    }
    return tmp;
}

int query (int l_root, int r_root, int k) {
    int l = 1, r = n;
    while (l < r) {
        int mid = (l+r)>>1;
        if (tree[tree[l_root].l].num - tree[tree[r_root].l].num >= k) {
            r = mid;
            l_root = tree[l_root].l;
            r_root = tree[r_root].l;
        }
        else {
            l = mid+1;
            k -= (tree[tree[l_root].l].num - tree[tree[r_root].l].num);
            l_root = tree[l_root].r;
            r_root = tree[r_root].r;
        }
    }
    return l;
}

int main () {
    //freopen ("in.txt", "r", stdin);
    int t;
    scanf ("%d", &t);
    while (t--) {
            scanf ("%d%d", &n, &q);
        num.clear ();
        gg.clear ();
        for (int i = 1; i <= n; i++) {
            scanf ("%d", &a[i]);
            num.push_back (a[i]);
        }
        init ();
        cnt = 0;
        T[n+1] = build_tree (1, n);
        for (int i = n; i >= 1; i--) {
            int pos = gg[a[i]];
            T[i] = update (T[i+1], pos, 1);
        }
        while (q--) {
            int l, r, k;
            scanf ("%d%d%d", &l, &r, &k);
            printf ("%d\n", p[query (T[l], T[r+1], k)]);
        }
    }
    return 0;
}
```

`SPOJ COT`:和数列的第k小类似不过写起来烦一点.在数列上求区间第k小是两个前缀减一减,树上就是u-root,v-root前缀加一加减去两倍LCA-root,新建线段树都是在父亲节点的基础上更新的.

```
#define maxn 211111
#define maxm maxn*40

int a[maxn], n, m, q;
vector <int> num;
map <int , int> gg;
struct E {
    int v, next;
}edge[maxn];
int head[maxn], cnt, Hash[maxn];

//主席树
struct node {
    int l, r, num;
}tree[maxm];
int tot, T[maxn];
int build_tree (int l, int r) {
    int root = tot++;
    tree[root].num = 0;
    if (l == r)
        return root;
    int mid = (l+r)>>1;
    tree[root].l = build_tree (l, mid);
    tree[root].r = build_tree (mid+1, r);
    return root;
}

int update (int root, int pos, int val) {
    int new_root = tot++;
    int tmp = new_root;
    tree[new_root].num = tree[root].num+val;
    int l = 1, r = m;
    while (l < r) {
        int mid = (l+r)>>1;
        if (pos <= mid) {
            tree[new_root].r = tree[root].r;
            tree[new_root].l = tot++;
            new_root = tree[new_root].l;
            root = tree[root].l;
            r = mid;
        }
        else {
            tree[new_root].l = tree[root].l;
            tree[new_root].r = tot++;
            new_root = tree[new_root].r;
            root = tree[root].r;
            l = mid+1;
        }
        tree[new_root].num = tree[root].num + val;
    }
    return tmp;
}

void dfs (int u, int fa) {
    T[u] = update (T[fa], a[u], 1);
    for (int i = head[u]; i != -1; i = edge[i].next) {
        int v = edge[i].v;
        if (v == fa) continue;
        dfs (v, u);
    }
}

void build () {
    tot = 0;
    T[n+1] = build_tree (1, m);
    dfs (1, 0);
```

```
}

int query (int l_root, int r_root, int root, int k) {//u,v,lca的主席树节点
    int pos = a[root];
    root = T[root];
    int l = 1, r = m;
    while (l < r) {
        int mid = (l+r)>>1;
        int tmp = tree[tree[l_root].l].num + tree[tree[r_root].l].num -
        tree[tree[root].l].num*2 + (pos >= l && pos <= mid);
        if (tmp >= k) {
            l_root = tree[l_root].l;
            r_root = tree[r_root].l;
            root = tree[root].l;
            r = mid;
        }
        else {
            k -= tmp;
            l_root = tree[l_root].r;
            r_root = tree[r_root].r;
            root = tree[root].r;
            l = mid+1;
        }
    }
    return l;
}

//LCA
int fa[maxn][22], deep[maxn];//节点i第2^j个祖先 深度
int DEG;
bool vis[maxn];

void bfs (int root) {
    DEG = 20;
    queue <int> q;
    while (!q.empty ()) q.pop ();
    deep[root] = 0;
    fa[root][0] = root;
    q.push (root);
    memset (vis, 0, sizeof vis); vis[root] = 1;
    while (!q.empty ()) {
        int u = q.front (); q.pop ();
        for (int i = 1; i < DEG; i++) {
            fa[u][i] = fa[fa[u][i-1]][i-1];
        }
        for (int i = head[u]; i != -1; i = edge[i].next) {
            int v = edge[i].v;
            if (vis[v]) continue;
            deep[v] = deep[u]+1;
            fa[v][0] = u;
            q.push (v);
            vis[v] = 1;
        }
    }
}

int LCA (int u, int v) {
    if (deep[u] > deep[v])
        swap (u, v);
    int hu = deep[u], hv = deep[v];
    int tu = u, tv = v;
    for (int det = hv-hu, i = 0; det; det >>= 1, i++) if (det&1) {
        tv = fa[tv][i];
    }
    if (tu == tv)
        return tu;
    for (int i = DEG-1; i >= 0; i--) {
        if (fa[tu][i] == fa[tv][i])
            continue;
        tu = fa[tu][i];
        tv = fa[tv][i];
    }
    return fa[tu][0];
}

void add_edge (int u, int v) {
    edge[cnt].v = v, edge[cnt].next = head[u], head[u] = cnt++;
}

void init () {//初始化
    gg.clear ();
    num.clear ();
    memset (head, -1, sizeof head);
    cnt = 0;
}

void lisan () {//离散化
    int cnt = 0;
    sort (num.begin (), num.end ());
    for (int i = 0; i < n; i++) {
        if (!i || num[i] != num[i-1]) {
            gg[num[i]] = ++cnt;
            Hash[cnt] = num[i];
        }
    }
    for (int i = 1; i <= n; i++) a[i] = gg[a[i]];
    m = cnt;
}

int main () {
    while (scanf ("%d%d", &n, &q) == 2) {
        init ();
        for (int i = 1; i <= n; i++) {
            scanf ("%d", &a[i]);
            num.push_back (a[i]);
        }
        lisan ();
        for (int i = 1; i < n; i++) {
            int u, v;
            scanf ("%d%d", &u, &v);
            add_edge (u, v);
            add_edge (v, u);
        }
        bfs (1);
        build ();
        for (int i = 1; i <= q; i++) {
            int u, v, k;
            scanf ("%d%d%d", &u, &v, &k);
            int fa = LCA (u, v);
            printf ("%d\n", Hash[query (T[u], T[v], fa, k)]);
        }
    }
    return 0;
}
```

字符串相关数据结构

哈希

```
const ull seed = 131;
ull hash (string s) {
    int n = s.length (); ull ans = 0;
    for (int i = 0;  i < n; i++) {
        ans = ans*seed+s[i]-'a'+1;
    }
    return ans;
}
```

KMP

- next数组的含义：next[i]表示str[0...i-1]最大的自匹配前缀和后缀的长度。

```
void get_next (int *p) {
    int t;
    t = next[0] = -1;
    int j = 0;
    while (j < m) {
        if (t < 0 || p[j] == p[t]) {//匹配
            j++, t++;
            next[j] = t;
        }
```

```
            else //失配
                t = next[t];
        }
    }

    void get_next (int *p) {
        int t;
        t = next[0] = -1;
        int j = 0;
        while (j < m) {
            if (t < 0 || p[j] == p[t]) {//匹配
                j++, t++;
                next[j] = (p[j] != p[t] ? t : next[t]);
            }
            else //失配
                t = next[t];
        }
    }
```

`HDU 1686`：文本串出现了多少次模式串(可以重叠)。（如果不能重叠在找到一次以后直接将j修改为0就可以）

```
    int kmp () {
        int ans = 0;
        get_next (P);
        int i = 0, j = 0;
        while (i < n && j < m) {
            if (j < 0 || T[i] == P[j]) {
                if (j == m-1) {
                    ans++;
                    j = next[j];
                    continue;
                }
                i++, j++;
            }
            else {
                j = next[j];
            }
        }
        return ans;
    }
```

`HDU 3746`：最少补充多少字符使原串变成至少循环两次的串。

假设原串长n,那么n-next[n]就是原串的一个最小循环节，然后讨论一下即可。

```
    void get_next (char *p) {
        int m = strlen (p);
        int t;
        t = next[0] = -1;
        int j = 0;
        while (j < m) {
            if (t < 0 || p[j] == p[t]) {//匹配
                j++, t++;
                next[j] = t;
            }
            else //失配
                t = next[t];
        }
    }

    int main () {
        int t;
        scanf ("%d", &t);
        while (t--) {
            scanf ("%s", T);
            n = strlen (T);
            get_next (T);
            int j = n;
            int len = j-next[j];//循环节的长度
            if (n%len == 0) {
                if (n/len > 1) printf ("0\n");
                else printf ("%d\n", len);
            }
            else
                printf ("%d\n", len-n%len);
        }
        return 0;
    }
```

最大(小)表示法

- 求一个循环字符串字典序最小(大)的字符下标

```
//flag=1表示最小表示 flag=0表示最大表示
//原字符串下标从0开始
int min_max_express (char *s, int len,bool flag) {
    int i=0,j=1,k=0;
    while (i<len && j<len && k<len) {
        int t = s[(j+k)%len]-s[(i+k)%len];
        if (t==0) k++;
        else {
            if (flag) {
                if (t>0) j+=k+1;
                else i+=k+1;
            }
            else {
                if (t>0) i+=k+1;
                else j+=k+1;
            }
            if (i==j) j++;
```

```
                k=0;
            }
        }
        return min(i,j);
    }
```

manachar

- len数组的含义：len[i]表示以s[i]为回文中心的的最长回文串的半径（至少为1）。

原串：s[0]s[1]s[2]...s[n-1] 新串:@#s[0]#s[1]#s[2]...#s[n-1]#~

```
int len[maxn<<1];
char a[maxn];

int manachar (char *p) {
    char s[maxn<<1];//构造新串
    int n = strlen (p), l = 0;
    s[l++] = '@';
    s[l++] = '#';
    for (int i = 0; i < n; i++) {
        s[l++] = p[i];
        s[l++] = '#';
    }
    s[l++] = '~'; s[l] = 0;

    int Max = 0, pos = 0, ans = 0;
    for (int i = 1; i < l; i++) {
        if (Max > i) {
            len[i] = min (len[2*pos-i], Max-i);
        }
        else
            len[i] = 1;
        while (s[i+len[i]] == s[i-len[i]])
            len[i]++;
        ans = max (ans, len[i]);
        if (len[i]+i > Max) {
            Max = len[i]+i;
            pos = i;
        }
    }
    return ans-1;
}
```

字典树

01字典树

```
struct Trie{
    LL ch[N][2],sz;
    int val[N],ed[N];
    void Init(){ memset(ch[0],0,sizeof(ch[0])); sz=1;}
    void Insert(LL x)
    {
        int u=0;
        for(int i=29;i>=0;i--)
        {
            int v=(x&(1ll<<i))>0?1:0;
            if(!ch[u][v])
            {
                memset(ch[sz],0,sizeof(ch[sz]));
                val[sz]=0;//每个节点的附加信息
                ed[sz]=0;
                ch[u][v]=sz++;//存储该节点的编号sz
            }
            u=ch[u][v];
            val[u]++;
        }
        ed[u]=x;//最后一个节点记录数值
    }
    void Delete(LL x)
    {
        int u=0;
        for(int i=29;i>=0;i--)
        {
            int v=(x&(1ll<<i))>0?1:0;
            u=ch[u][v];
            val[u]--;
        }
        //ed[u]=0;//可能有重复数字，若置为0后该位置的另一个数字也被删除了
    }
    int Search(LL x)
    {
        int u=0;
        for(int i=29;i>=0;i--)
        {
            int v=(x&(1ll<<i))>0;//判断条件
            if(ch[u][!v]&&val[ch[u][!v]])
            {
                u=ch[u][!v];
            }
            else u=ch[u][v];
        }
        return x^ed[u];
    }
}trie;
```

AC自动机

本质上在字典树中增加了一系列失配指针。**注意在询问时是否需要沿着失配边走。**

```
//查询文本串中出现了多少模式串
struct trie {
    int next[maxn][26], fail[maxn], end[maxn];
    int root, cnt;
    int new_node () {
        memset (next[cnt], -1, sizeof next[cnt]);
        end[cnt++] = 0;
        return cnt-1;
    }
    void init () {
        cnt = 0;
        root = new_node ();
    }
    void insert (char *buf) {//字典树插入一个单词
        int len = strlen (buf);
        int now = root;
        for (int i = 0; i < len; i++) {
            int id = buf[i]-'a';
            if (next[now][id] == -1) {
                next[now][id] = new_node ();
            }
            now = next[now][id];
        }
        end[now]++;
    }
    void build () {//构建fail指针
        queue <int> q;
        fail[root] = root;
        for (int i = 0; i < 26; i++) {
            if (next[root][i] == -1) {
                next[root][i] = root;
            }
            else {
                fail[next[root][i]] = root;
                q.push (next[root][i]);
            }
        }
        while (!q.empty ()) {
            int now = q.front (); q.pop ();
            for (int i = 0; i < 26; i++) {
                if (next[now][i] == -1) {
                    next[now][i] = next[fail[now]][i];
                }
                else {
                    fail[next[now][i]] = next[fail[now]][i];
                    q.push (next[now][i]);
                }
            }
        }
    }
    int query (char *buf) {
        int len = strlen (buf);
        int now = root;
        int res = 0;
        for (int i = 0; i < len; i++) {
            int id = buf[i]-'a';
            now = next[now][id];
            int tmp = now;
            while (tmp != root) {
                if (end[tmp]) {
                    res += end[tmp];
                    end[tmp] = 0;
                }
                tmp = fail[tmp];//沿着失配边走
            }
        }
        return res;
    }
}ac;
```

- 字符串之间有依赖关系的DP，很多可以转化成AC自动机节点图上的DP，一维记录当前在AC自动机的哪一个节点。

后缀数组

$sa[i]$表示字典序第i大的后缀下标(字典序排名依次是$1-len(string)$); $rank[i]$表示下标为i的后缀字典序排名; $height[i]$表示$sa[i]$和$sa[i-1]$最长公共前缀的长度.

- 一个性质: $LCP(suffix[i], suffix[j]) = min\left\{height[i+1], height[i+2] \ldots height[j]\right\} (rank[i] < rank[j])$
- 倍增法求后缀数组(记得在原串末尾+0)

```
//m表示字符中最大的值
int t1[maxn],t2[maxn],c[maxn];
bool cmp(int *r,int a,int b,int l)
{
    return r[a] == r[b] && r[a+l] == r[b+l];
}
void da(int str[],int sa[],int rank[],int height[],int n,int m)
{
    n++;
    int i, j, p, *x = t1, *y = t2;
    //第一轮基数排序，如果s的最大值很大，可改为快速排序
    for(i = 0; i < m; i++)c[i] = 0;
    for(i = 0; i < n; i++)c[x[i] = str[i]]++;
    for(i = 1; i < m; i++)c[i] += c[i-1];
    for(i = n-1; i >= 0; i--)sa[--c[x[i]]] = i;
    for(j = 1; j <= n; j <<= 1)
    {
        p = 0;
        //直接利用sa数组排序第二关键字
        for(i = n-j; i < n; i++)y[p++] = i;//后面的j个数第二关键字为空的最小
        for(i = 0; i < n; i++)if(sa[i] >= j)y[p++] = sa[i] - j;
        //这样数组y保存的就是按照第二关键字排序的结果
        //基数排序第一关键字
        for(i = 0; i < m; i++)c[i] = 0;
        for(i = 0; i < n; i++)c[x[y[i]]]++;
        for(i = 1; i < m; i++)c[i] += c[i-1];
        for(i = n-1; i >= 0; i--)sa[--c[x[y[i]]]] = y[i];
        //根据sa和x数组计算新的x数组
        swap(x,y);
        p = 1;
        x[sa[0]] = 0;
        for(i = 1; i < n; i++)
            x[sa[i]] = cmp(y,sa[i-1],sa[i],j)?p-1:p++;
        if(p >= n)break;
        m = p;//下次基数排序的最大值
    }
    int k = 0;
    n--;
    for(i = 0; i <= n; i++)rank[sa[i]] = i;
    for(i = 0; i < n; i++)
    {
        if(k)   k--;
        j = sa[rank[i]-1];
        while(str[i+k] == str[j+k])
            k++;
        height[rank[i]] = k;
    }
}
```

`POJ 1743`：求两个最长的不重叠子串, 满足两个串对应下标的差值相等. 做法：对于相邻的两个数直接做差得到一个新串, 直接对新串求最长不重叠子串. 非常经典的做法, 二分长度, 然后对height数组分组, 满足这个长度的分成一组, 然后判断组中下标最大最小之差.

```
#define maxn 20005
using namespace std;
int t1[maxn],t2[maxn],c[maxn];
bool cmp(int *r,int a,int b,int l)
{
    return r[a] == r[b] && r[a+l] == r[b+l];
}
void da(int str[],int sa[],int rank[],int height[],int n,int m)
{
    n++;
    int i, j, p, *x = t1, *y = t2;
    //第一轮基数排序，如果s的最大值很大，可改为快速排序
    for(i = 0; i < m; i++)c[i] = 0;
    for(i = 0; i < n; i++)c[x[i] = str[i]]++;
    for(i = 1; i < m; i++)c[i] += c[i-1];
    for(i = n-1; i >= 0; i--)sa[--c[x[i]]] = i;
    for(j = 1; j <= n; j <<= 1)
    {
        p = 0;
        //直接利用sa数组排序第二关键字
        for(i = n-j; i < n; i++)y[p++] = i;//后面的j个数第二关键字为空的最小
        for(i = 0; i < n; i++)if(sa[i] >= j)y[p++] = sa[i] - j;
        //这样数组y保存的就是按照第二关键字排序的结果
        //基数排序第一关键字
        for(i = 0; i < m; i++)c[i] = 0;
        for(i = 0; i < n; i++)c[x[y[i]]]++;
        for(i = 1; i < m; i++)c[i] += c[i-1];
        for(i = n-1; i >= 0; i--)sa[--c[x[y[i]]]] = y[i];
        //根据sa和x数组计算新的x数组
        swap(x,y);
        p = 1;
        x[sa[0]] = 0;
        for(i = 1; i < n; i++)
            x[sa[i]] = cmp(y,sa[i-1],sa[i],j)?p-1:p++;
        if(p >= n)break;
        m = p;//下次基数排序的最大值
    }
    int k = 0;
    n--;
    for(i = 0; i <= n; i++)rank[sa[i]] = i;
    for(i = 0; i < n; i++)
    {
        if(k)   k--;
        j = sa[rank[i]-1];
        while(str[i+k] == str[j+k])
            k++;
        height[rank[i]] = k;
    }
}
int rank[maxn], height[maxn];
int str[maxn];
int sa[maxn];
int n;
#define INF 111111

bool ok (int x) {
    int Min = INF, Max = 0;
    for (int i = 1; i <= n; i++) {
        if (height[i] >= x) {
            Min = min (Min, sa[i]);
            Max = max (Max, sa[i]);
        }
        else {
            if (Max-Min >= x)
                return 1;
            Max = sa[i];
```

```
                Min = sa[i];
            }
        }
        return Max-Min >= x;
    }

    int solve () {
        int l = 0, r = n/2;
        while (r-l > 1) {
            int mid = (r+l)>>1;
            if (ok (mid)) l = mid;
            else r = mid;
        }
        return (ok (r) ? r : l);
    }

    int main(){
        while (scanf ("%d", &n) == 1 && n) {
            for (int i = 0; i < n; i++) {
                scanf ("%d", &str[i]);
            }
            if (n <= 9) {
                printf ("0\n");
                continue;
            }
            n--;
            for (int i = 0; i < n; i++) {
                str[i] = 100+str[i+1]-str[i];
            }
            str[n] = 0;
            da(str, sa, rank, height, n+1, 188);
            int ans = solve ()+1;
            printf ("%d\n", (ans >= 5 ? ans : 0));
        }
        return 0;
    }
```

`POJ 3261`：求重复k次的最长子串 解法：二分结果, 按照height分组, 判断是不是有大于k的组.

```
    #define maxn 200005
    using namespace std;
    int t1[maxn],t2[maxn],c[maxn];
    bool cmp(int *r,int a,int b,int l)
    {
        return r[a] == r[b] && r[a+l] == r[b+l];
    }
    void da(int str[],int sa[],int rank[],int height[],int n,int m)
    {
        n++;
        int i, j, p, *x = t1, *y = t2;
        //第一轮基数排序，如果s的最大值很大，可改为快速排序
        for(i = 0; i < m; i++)c[i] = 0;
        for(i = 0; i < n; i++)c[x[i] = str[i]]++;
        for(i = 1; i < m; i++)c[i] += c[i-1];
        for(i = n-1; i >= 0; i--)sa[--c[x[i]]] = i;
        for(j = 1; j <= n; j <<= 1)
        {
            p = 0;
            //直接利用sa数组排序第二关键字
            for(i = n-j; i < n; i++)y[p++] = i;//后面的j个数第二关键字为空的最小
            for(i = 0; i < n; i++)if(sa[i] >= j)y[p++] = sa[i] - j;
            //这样数组y保存的就是按照第二关键字排序的结果
            //基数排序第一关键字
            for(i = 0; i < m; i++)c[i] = 0;
            for(i = 0; i < n; i++)c[x[y[i]]]++;
            for(i = 1; i < m; i++)c[i] += c[i-1];
            for(i = n-1; i >= 0; i--)sa[--c[x[y[i]]]] = y[i];
            //根据sa和x数组计算新的x数组
            swap(x,y);
            p = 1;
            x[sa[0]] = 0;
            for(i = 1; i < n; i++)
                x[sa[i]] = cmp(y,sa[i-1],sa[i],j)?p-1:p++;
            if(p >= n)break;
            m = p;//下次基数排序的最大值
        }
        int k = 0;
        n--;
        for(i = 0; i <= n; i++)rank[sa[i]] = i;
        for(i = 0; i < n; i++)
        {
            if(k)   k--;
            j = sa[rank[i]-1];
            while(str[i+k] == str[j+k])
                k++;
            height[rank[i]] = k;
        }
    }
    int rank[maxn], height[maxn];
    int str[maxn];
    int sa[maxn];
    int n, k;

    bool ok (int x) {
        int ans = 1;
        for (int i = 2; i <= n; i++) {
            if (height[i] >= x) {
                ans++;
                if (ans >= k)
                    return 1;
            }
            else
                ans = 1;
        }
        return 0;
    }

    int solve () {
        int l = 0, r = n;
        while (r-l > 1) {
            int mid = (l+r) >>1;
            if (ok (mid)) l = mid;
            else r=  mid;
        }
        return (ok (r) ? r : l);
    }


    int cnt, num[maxn], gg[maxn];
    int lisanhua () {
        cnt = 0;
        for (int i = 0; i < n; i++) num[i] = i;
        sort (num, num+n);
        for (int i = 0; i < n; i++) if (!i || num[i] != num[i-1])
            gg[cnt++] = num[i];
        for (int i = 0; i < n; i++)
            str[i] = lower_bound (gg, gg+cnt, str[i])-gg+1;
        return cnt+1;
    }

    int main(){
        while (cin >> n >> k) {
            int Max = 0;
            for (int i = 0; i < n; i++) {
                cin >> str[i];
                Max = max (Max, str[i]);
            }
            str[n] = 0;
            int m = lisanhua ();
            da (str, sa, rank, height, n, m+2);
            int ans = solve ();
            cout << ans << endl;
        }
        return 0;
    }
```

`SPOJ 694`: 不重复子串的个数 根据sa数组和height数组的含义, sa[i]后缀总共有n−sa[i]个前缀, 有height[i]个前缀和之前的重复, 所以要减去. 最后答案是$\sum_{i=1}^{n} -sa[i] - eight[i]$.

```
#define maxn 200005
using namespace std;
```

```
int t1[maxn],t2[maxn],c[maxn];
bool cmp(int *r,int a,int b,int l)
{
    return r[a] == r[b] && r[a+l] == r[b+l];
}
void da(int str[],int sa[],int rank[],int height[],int n,int m)
{
    n++;
    int i, j, p, *x = t1, *y = t2;
    //第一轮基数排序，如果s的最大值很大，可改为快速排序
    for(i = 0; i < m; i++)c[i] = 0;
    for(i = 0; i < n; i++)c[x[i] = str[i]]++;
    for(i = 1; i < m; i++)c[i] += c[i-1];
    for(i = n-1; i >= 0; i--)sa[--c[x[i]]] = i;
    for(j = 1; j <= n; j <<= 1)
    {
        p = 0;
        //直接利用sa数组排序第二关键字
        for(i = n-j; i < n; i++)y[p++] = i;//后面的j个数第二关键字为空的最小
        for(i = 0; i < n; i++)if(sa[i] >= j)y[p++] = sa[i] - j;
        //这样数组y保存的就是按照第二关键字排序的结果
        //基数排序第一关键字
        for(i = 0; i < m; i++)c[i] = 0;
        for(i = 0; i < n; i++)c[x[y[i]]]++;
        for(i = 1; i < m; i++)c[i] += c[i-1];
        for(i = n-1; i >= 0; i--)sa[--c[x[y[i]]]] = y[i];
        //根据sa和x数组计算新的x数组
        swap(x,y);
        p = 1;
        x[sa[0]] = 0;
        for(i = 1; i < n; i++)
            x[sa[i]] = cmp(y,sa[i-1],sa[i],j)?p-1:p++;
        if(p >= n)break;
        m = p;//下次基数排序的最大值
    }
    int k = 0;
    n--;
    for(i = 0; i <= n; i++)rank[sa[i]] = i;
    for(i = 0; i < n; i++)
    {
        if(k)   k--;
        j = sa[rank[i]-1];
        while(str[i+k] == str[j+k])
            k++;
        height[rank[i]] = k;
    }
}
int rank[maxn], height[maxn];
int str[maxn];
char s[maxn];
int sa[maxn];
int n, k;

int main(){
    ios::sync_with_stdio(0);
    int t;
    cin >> t;
    while (t--) {
        cin >> s;
        n = strlen (s);
        for (int i = 0; i < n; i++) str[i] = s[i];
        str[n] = 0;
        da (str, sa, rank, height, n, 233);
        long long ans = 0;
        for (int i = 1; i <= n; i++) {
            ans += n-sa[i]-height[i];
        }
        cout << ans << endl;
    }
    return 0;
}
```

`POJ 2774`: 最长公共子串(nlgn) 把第二个串放到第一个串的后面, 中间用一个失配符隔开, 然后遍历height数组维护最大子串长度. 要避免出现在同一串中的公共子串.

```
#define maxn 200005
using namespace std;
int t1[maxn],t2[maxn],c[maxn];
bool cmp(int *r,int a,int b,int l)
{
    return r[a] == r[b] && r[a+l] == r[b+l];
}
void da(int str[],int sa[],int rank[],int height[],int n,int m)
{
    n++;
    int i, j, p, *x = t1, *y = t2;
    //第一轮基数排序，如果s的最大值很大，可改为快速排序
    for(i = 0; i < m; i++)c[i] = 0;
    for(i = 0; i < n; i++)c[x[i] = str[i]]++;
    for(i = 1; i < m; i++)c[i] += c[i-1];
    for(i = n-1; i >= 0; i--)sa[--c[x[i]]] = i;
    for(j = 1; j <= n; j <<= 1)
    {
        p = 0;
        //直接利用sa数组排序第二关键字
        for(i = n-j; i < n; i++)y[p++] = i;//后面的j个数第二关键字为空的最小
        for(i = 0; i < n; i++)if(sa[i] >= j)y[p++] = sa[i] - j;
        //这样数组y保存的就是按照第二关键字排序的结果
        //基数排序第一关键字
        for(i = 0; i < m; i++)c[i] = 0;
        for(i = 0; i < n; i++)c[x[y[i]]]++;
        for(i = 1; i < m; i++)c[i] += c[i-1];
        for(i = n-1; i >= 0; i--)sa[--c[x[y[i]]]] = y[i];
        //根据sa和x数组计算新的x数组
        swap(x,y);
        p = 1;
        x[sa[0]] = 0;
        for(i = 1; i < n; i++)
            x[sa[i]] = cmp(y,sa[i-1],sa[i],j)?p-1:p++;
        if(p >= n)break;
        m = p;//下次基数排序的最大值
    }
    int k = 0;
    n--;
    for(i = 0; i <= n; i++)rank[sa[i]] = i;
    for(i = 0; i < n; i++)
    {
        if(k)   k--;
        j = sa[rank[i]-1];
        while(str[i+k] == str[j+k])
            k++;
        height[rank[i]] = k;
    }
}
int rank[maxn], height[maxn];
int str[maxn];
char s1[maxn], s2[maxn];
int sa[maxn];
int n, m, len;

bool legal (int i, int j) {
    if (i > j) swap (i, j);
    return (i < n && j > n);
}

void solve () {
    int Max = 0;
    for (int i = 2; i <= len; i++) {
        if (height[i] >= Max && legal (sa[i], sa[i-1]))
            Max = height[i];
    }
    cout << Max << endl;
}

int main(){
    ios::sync_with_stdio(0);
    while (cin >> s1 >> s2) {
        n = strlen (s1), m = strlen (s2);
        for (int i = 0; i < n; i++) str[i] = s1[i];
        str[n] = 1;
        for (int i = n+1; i <= n+m; i++) str[i] = s2[i-n-1];
        len = n+m+1;
        str[len] = 0;
        da (str, sa, rank, height, len, 233);
        solve ();
    }
    return 0;
}
```

基本子串字典(HDU 5814)

- nlgn预处理字符串，lgn^2求某一个子串的最小循环节

```
const int N = 100000 + 10;
const int M = 17;

char S[N];
int dict[M][N];
int sz[M];
int cnt[N], tmp1[N], tmp2[N];
vector<int> A[M][N];
int d, k, cn;

inline int mylog(int x) {
    int i, j;
    for (i=1,j=0; i<x; i*=2,j++);
    return -- j;
}

inline int next(int d, const vector<int>& V, int x) {
    int y;
    y = lower_bound(V.begin(), V.end(), x + 1) - V.begin();
    if (y < V.size()) {
        y = V[y];
        if (y <= x + (1<<d)) return y;
    }
    return -1;
}

inline int prev(int d, const vector<int>& V, int x) {
    int y;
    y = lower_bound(V.begin(), V.end(), x) - V.begin();
    if (y > 0) {
        y = V[y-1];
        if (y >= x - (1<<d)) return y;
    }
    return -1;
}

inline int intersect(const int A[], int n, const int B[], int m) {
    int i, j;
    for (i=n-1; i>=0; i--) for (j=m-1; j>=0; j--) if (A[i] == B[j]) return A[i];
    return -1;
}

inline int intersect(const int A[], int n, int a, int d, int b) {
    int i;
    for (i=n-1; i>=0; i--) if ((A[i] - a) % d == 0 && A[i] >= a && A[i] <= b) return
A[i];
    return -1;
}

inline int intersect(int a1, int b1, int a2, int b2, int d) {
    if (b1 < a2 || b2 < a1) return -1;
    return min(b1, b2);
}

inline void read_int (int& x) {
    char c;
    while (c = getchar(), c < '0' || c > '9');
    x = c - '0';
    while (c = getchar(), c >= '0' && c <= '9') x = x * 10 + c - '0';
}

int main() {
    int res, n, q, t, len, s, h, a, b, c, nxt, prv, n1, n2, i, j;
    int P[5], Q[5];
    scanf("%d", &t);
    cn = 0;
    while (t --) {
        printf("Case #%d:\n", ++ cn);
        scanf("%s", S);
        n = strlen(S);
        for (i=0; i<n; i++) {
            dict[0][i] = S[i] - 'a' + 1;
            A[0][dict[0][i]].push_back(i);
        }
        sz[0] = 26;
        for (d=0,k=1; k*2<=n; d++,k*=2) {
            s = n - k * 2;
            memset(cnt, 0, sizeof(cnt[0]) * (sz[d] + 1));
            for (i=0; i<=s; i++) cnt[dict[d][i]] ++;
            for (i=2; i<=sz[d]; i++) cnt[i] += cnt[i-1];
            for (i=0; i<=s; i++) tmp1[cnt[dict[d][i]-1]++] = i;
            memset(cnt, 0, sizeof(cnt[0]) * (sz[d] + 1));
            for (i=0; i<=s; i++) cnt[dict[d][i+k]] ++;
            for (i=2; i<=sz[d]; i++) cnt[i] += cnt[i-1];
            for (i=0; i<=s; i++) tmp2[cnt[dict[d][tmp1[i]+k]-1]++] = tmp1[i];
            dict[d+1][tmp2[0]] = 1;
            A[d+1][dict[d+1][tmp2[0]]].push_back(tmp2[0]);
            for (i=1; i<=s; i++) {
                dict[d+1][tmp2[i]] = dict[d+1][tmp2[i-1]] + (dict[d][tmp2[i]] != 
dict[d][tmp2[i-1]] || dict[d][tmp2[i]+k] != dict[d][tmp2[i-1]+k]);
                A[d+1][dict[d+1][tmp2[i]]].push_back(tmp2[i]);
            }
            sz[d+1] = dict[d+1][tmp2[s]];
        }
        h = d;
        scanf("%d", &q);
        while (q --)  {
            read_int(a);
            read_int(b);
            a --;
            b --;
            len = b - a + 1;
            if (len == 1) {
                printf("1\n");
                continue;
            }
            d = mylog(len);
            nxt = next(d, A[d][dict[d][a]], a);
            if (nxt != -1 && nxt <= b - (1<<d) + 1) {
                prv = prev(d, A[d][dict[d][b-(1<<d)+1]], b - (1<<d) + 1);
                if (prv != -1 && prv >= a && nxt - a == b - (1<<d) + 1 - prv) {
                    printf("%d\n", nxt - a);
                    continue;
                }
            }
            res = 0;
            for (--d,k=(1<<d); d>=0; d--,k/=2) {
                c = b - k + 1;
                P[0] = a;
                for (n1=1; n1<=3; n1++) {
                    P[n1] = next(d, A[d][dict[d][c]], P[n1-1]);
                    if (P[n1] == -1 || P[n1] - P[0] > k) break;
                }
                n1 --;
                Q[0] = c;
                for (n2=1; n2<=3; n2++) {
                    Q[n2] = prev(d, A[d][dict[d][a]], Q[n2-1]);
                    if (Q[n2] == -1 || Q[0] - Q[n2] > k) break;
                }
                n2 --;
                if (n1 < 1 || n2 < 1) continue;
                for (j=1; j<=n1; j++) P[j] = P[j] - a + k;
                for (j=1; j<=n2; j++) Q[j] = b - Q[j] + 1;
                if (n1 > 2) {
                    P[n1+1] = prev(d, A[d][dict[d][c]], a + k + 1);
                    P[n1+1] = P[n1+1] - a + k;
                }
                if (n2 > 2) {
                    Q[n2+1] = next(d, A[d][dict[d][a]], b - 2 * k);
```

```
                        Q[n2+1] = b - Q[n2+1] + 1;
                    }
                    if (n1 <= 2 && n2 <= 2) res = max(res, intersect(P + 1, n1, Q + 1, 
n2));
                    else if (n1 <= 2 && n2 > 2) res = max(res, intersect(P + 1, n1, Q[1], 
Q[2] - Q[1], Q[n2+1]));
                    else if (n1 > 2 && n2 <= 2) res = max(res, intersect(Q + 1, n2, P[1], 
P[2] - P[1], P[n1+1]));
                    else res = max(res, intersect(P[1], P[n1+1], Q[1], Q[n2+1], P[2] - 
P[1]));
                    if (res) break;
                }
                if (S[a] == S[b] && !res) res = 1;
                printf("%d\n", len - res);
            }
            for (i=0; i<=h; i++) for (j=0; j<=sz[i]; j++) A[i][j].clear();
        }
        return 0;
}
```

动态规划

普通DP

有趣的转移

`POJ 3783`: n层楼m个鸡蛋最坏的情况下最少多少次确定哪个楼层鸡蛋掉下去刚好摔碎. 设dp[i][j]表示i层楼j个鸡蛋最坏情况下多少次确定,那么前一次扔鸡蛋的楼层假设是k,如果摔碎:dp[k-1][j-1],如果不碎:dp[i-k][j],因为要最坏的情况,所以取两者的较多者.dp[i][j]=min (max (dp[k-1][j-1], dp[i-k][j]))+1.

```
#define INF 1111111

int dp[1111][55];
int n, m;

int main () {
    //freopen ("in.txt", "r", stdin);
    int id, n, m;
    n = 1000, m = 50;
    for (int i = 1; i <= m; i++) dp[1][i] = 0;
    for (int i = 2; i <= n; i++) {
        dp[i][0] = INF;
        dp[i][1] = i-1;
    }
    for (int i = 2; i <= n; i++) {
        for (int j = 2; j <= m; j++)
            dp[i][j] = INF;
    }
    for (int i = 2; i <= n; i++) {
        for (int j = 2; j <= m; j++) {
            int cur = INF;
            for (int k = 1; k < i; k++) {
                cur = min (cur, max (dp[i-k][j], dp[k][j-1]));//没有破 破掉
            }
            dp[i][j] = min (dp[i][j], cur+1);
        }
    }
    int t;
    scanf ("%d", &t);
    while (t--) {
        scanf ("%d%d%d", &id, &m, &n);
        printf ("%d %d\n", id, dp[n][m]);
    }
    return 0;
}
```

`HDU 5550`:有n层楼，每层有两类人，假设是0，1，然后为每一层楼设置一个01属性使得所有人走到自己属性的楼层的距离和最小。 容易想到的是dp[i][0/1]表示这一层属性是0/1的情况下1-i层的最小值，但是有个问题就是另一种属性不知道应该去那一层，所以状态转变一下。因为层的属性肯定是连续的0或者1拼接成的，可以设dp[i][0/1]表示这一层属性是0/1的情况并且和前一层状态不同，1-i层的最小值。对于某一楼i假如是0，那么它之前肯定有一段1，然后就枚举这个1的状态转变楼层j，那么j-1必然是0，然后转移的时候，ij之间的1属性的人只需要i层里面的往下走一楼，0属性的人可以分成两半，前一半走到i，后一半走到j-1。另一种情况也类似。然后需要注意的是一个区间的人全部走到某一层。容易搞的是开一个二维数组，然而MLE。那就只有本方法了，各种前缀和弄起来

```
#define maxn 4111
#define INF 1111111111111111111

long long dp[maxn][2];
long long sum1[maxn][2], sum2[maxn][2];//0表示到n 1表示到1
long long all1[maxn], all2[maxn];//前缀和
long long a[maxn], b[maxn];
int n;

long long get_up (int i, int j, bool op) {//向上都跑到i
    if (i == j) return 0;
    long long ans = 0;
    if (!op) {
        long long tmp = (j == 1 ? all1[i-1] : all1[i-1]-all1[j-1]);
        long long cur = sum1[j][0]-sum1[i][0];
        ans = cur-tmp*(n-i);
    }
    else {
        long long tmp = (j == 1 ? all2[i-1] : all2[i-1]-all2[j-1]);
        long long cur = sum2[j][0]-sum2[i][0];
        ans = cur-tmp*(n-i);
    }
    return ans;
}
```

```
long long get_down (int i, int j, int op) {//向下都跑到i
    if (i == j) return 0;
    long long ans = 0;
    if (!op) {
        long long tmp = all1[j]-all1[i];
        long long cur = sum1[j][1]-sum1[i][1];
        ans = cur-tmp*(i-1);
    }
    else {
        long long tmp = all2[j]-all2[i];
        long long cur = sum2[j][1]-sum2[i][1];
        ans = cur-tmp*(i-1);
    }
    return ans;
}

int main () {
    //freopen ("in.txt", "r", stdin);
    int t, kase = 0;
    scanf ("%d", &t);
    while (t--) {
        scanf ("%d", &n);
        all1[0] = all2[0] = 0;
        for (int i = 1; i <= n; i++) {
            scanf ("%lld%lld", &a[i], &b[i]);
            all1[i] = all1[i-1]+a[i];
            all2[i] = all2[i-1]+b[i];
        }
        for (int i = n; i >= 1; i--) {
            if (i == n)
                sum1[i][0] = sum2[i][0] = 0;
            else {
                sum1[i][0] = sum1[i+1][0]+a[i]*(n-i);
                sum2[i][0] = sum2[i+1][0]+b[i]*(n-i);
            }
        }
        for (int i = 1; i <= n; i++) {
            if (i == 1)
                sum1[i][1] = sum2[i][1] = 0;
            else {
                sum1[i][1] = sum1[i-1][1]+a[i]*(i-1);
                sum2[i][1] = sum2[i-1][1]+b[i]*(i-1);
            }
        }
        for (int i = 2; i <= n; i++) {
            dp[i][0] = get_up (i, 1, 0)+b[i];
            dp[i][1] = get_up (i, 1, 1)+a[i];
        }
        for (int i = 3; i <= n; i++) {
            for (int j = i-1; j > 1; j--) {
                double mid = (i+j-1)/2.0;
                int up = ceil (mid), down = floor (mid);
                if (up == down)
                    up++;
                dp[i][0] = min (dp[i][0], dp[j][1]+b[i]+get_up (i, up, 0)+get_down
(j-1, down, 0)-a[j]);
                dp[i][1] = min (dp[i][1], dp[j][0]+a[i]+get_up (i, up, 1)+get_down
(j-1, down, 1)-b[j]);
            }
        }

        long long ans = INF;
        for (int i = 2; i <= n; i++) {
            ans = min (ans, dp[i][0]+get_down (i-1, n, 1)-b[i]);
            ans = min (ans, dp[i][1]+get_down (i-1, n, 0)-a[i]);
        }
        printf ("Case #%d: %lld\n", ++kase, ans);
    }
    return 0;
}
```

`codeforces 840C` :求相同数字不能相邻的排列方式。 预处理所有相同的块，然后按照块分配，用排列组合方法更新DP值。

```
int num[maxn], sum[maxn], cnt; //每个块的大小 块的前缀和 块的数量
long long dp[maxn][maxn]; //处理完第i个块 有j个不合法相邻对 有sum[i]-j-1个合法相邻对
void solve () {
    init ();
    Clear (dp, 0);
    dp[0][num[0]-1] = A[num[0]];
    for (int i = 1; i < cnt; i++) { //枚举当前处理的块
        for (int j = 0; j < sum[i-1]; j++) { // 枚举前面的块处理完还有j个不合法对 sum[i-
1]-j-1个合法对
            int p = num[i];
            for (int k = 1; k <= p; k++) { //枚举现在的块分成了k段 新增p-k个不合法对
                for (int l = 0; l <= min (j, k); l++) { //枚举消消除原来的l个不合法对
                    dp[i][j-l+p-k] += dp[i-1][j]*A[p]%mod *C[p-1][k-1]%mod *C[j]
[l]%mod *C[sum[i-1]-j+1][k-l]%mod;
                    dp[i][j-l+p-k] %= mod;
                }
            }
        }
    }
    printf ("%lld\n", dp[cnt-1][0]);
}
```

字典树DP

`NEU 1007`: 把一个字符串用几个子串表示,求最大的权值. 设dp(i)表示i位之前的字符串都处理完的最大权值,那么dp(i+len[j])=max (dp(i+len[j]), dp(i)+val[j]),其中i+1~i+len[j]这一段等于子串j,那么就可以用字典树存下原本所有的子串,然后每次从i开始从字典树的上面往下扫就可以了.

```
#define maxn 1111

struct node {
    int next[33];
    long long num;
}tree[31111];
int cnt;
char a[11111], s[33];
int n;
long long val;
long long dp[11111];

void init () {
    memset (tree, -1, sizeof tree);
    cnt = 0;
}

void add () {
    int l = strlen (s);
    int p = 0;
    for (int i = 0; i < l; i++) {
        int id = s[i]-'a';
        if (tree[p].next[id] == -1) {
            tree[p].next[id] = ++cnt;
        }
        p = tree[p].next[id];
    }
    tree[p].num = max (tree[p].num, val);
}

int main () {
    //freopen ("in.txt", "r", stdin);
    while (scanf ("%d%s", &n, a+1) == 2) {
        a[0] = '.';
        init ();
        for (int i = 0; i < n; i++) {
            scanf ("%s%lld", s, &val);
            add ();
        }
        memset (dp, -1, sizeof dp);
        dp[0] = 0;
        int l = strlen (a);
        l--;
        for (int i = 0; i < l; i++) {
            if (dp[i] == -1)
                continue;
            int p = 0;
            for (int j = i+1; j <= l; j++) {
                int id = a[j]-'a';
                if (tree[p].next[id] == -1)
                    break;
                if (tree[p].next[id] != -1 && tree[tree[p].next[id]].num != -1) {
                    dp[j] = max (dp[j], dp[i]+tree[tree[p].next[id]].num);
                }
                p = tree[p].next[id];
            }
        }
        printf ("%lld\n", dp[l] == 0 ? -1 : dp[l]);
    }
    return 0;
}
```

AC自动机DP

`HDU 3341` : :给出n个DNA模式串, 和一个文本串, 调整文本串字符的顺序使得所有模式串出现的次数和最多. 因为四种字符出现的次数都有限, 可以用类似于进制的做法记录字符出现的状态, 如果四个字符的总数有a, b, c, d, 当前状态出现的次数是x, y, z, k, 那么状态就记录为x×(b+1)×(c+1)×(d+1)+y×(c+1)×(d+1)+z×(d+1)+k×1,然后建立AC自动机在上面转移状态就好了.

```
int dp[maxn][11*11*11*11];//在字典树i节点 四种字母出现状态
int bit[5];
int query (int a, int b, int c, int d, char *buf) {
    memset (dp, -1, sizeof dp);
    dp[0][0] = 0;
    bit[0] = (b+1)*(c+1)*(d+1), bit[1] = (c+1)*(d+1), bit[2] = (d+1), bit[3] = 1;
    for (int x = 0; x <= a; x++) {
        for (int y = 0; y <= b; y++) {
            for (int z = 0; z <= c; z++) {
                for (int k = 0; k <= d; k++) {
                    int p = x*bit[0]+y*bit[1]+z*bit[2]+k*bit[3];
                    for (int i = 0; i < cnt; i++) {
                        if (dp[i][p] == -1)
                            continue;
                        for (int id = 0; id < 4; id++) {
                            if (id == 0 && x == a) continue;
                            if (id == 1 && y == b) continue;
                            if (id == 2 && z == c) continue;
                            if (id == 3 && k == d) continue;
                            int nexti = next[i][id];
                            int nextp = p + bit[id];
                            dp[nexti][nextp] = max (dp[nexti][nextp], dp[i][p] +
end[nexti]);
                        }
                    }
                }
            }
        }
    }
    int state = a*bit[0] + b*bit[1] + c*bit[2] + d*bit[3];
    int ans = 0;
    for (int i = 0; i < cnt; i++) ans = max (ans, dp[i][state]);
    return ans;
}
```

`HDU 2457`: 给定一个DNA串, 至少改变多少字母才能使得所有的模式串都不出现在这个DNA串里. 先给所有的模式串建立AC自动机,然后DNA串就不能走到自动机上的危险节点, 也就是模式串的结尾节点. 假设$dp[i][j]$表示在字典树上i节点, DNA串长为j的最小改变个数, 转移就是

$$d[i][j] = min\begin{cases} dp[k][j-1] & buf[k] = buf[i+1], k\text{是文本串}i\text{的后继不是模式串结尾} \\ dp[k][j-1]+1 & buf[k] \neq buf[i+1], k\text{不是文本串}i\text{的后继且}k\text{不是模式串结尾} \end{cases}$$

```
#define maxn 1005
#define INF 1111111

int dp[maxn][maxn];//在字典树上i节点 长度为j 最少改变了多少次
int n;
char str[maxn];

void change (char *s) {
    int len = strlen (s);
    for (int i = 0; i < len; i++) {
        if (s[i] == 'A') s[i] = 'a';
        else if (s[i] == 'C') s[i] = 'b';
        else if (s[i] == 'G') s[i] = 'c';
        else s[i] = 'd';
    }
}

struct trie {
    int next[maxn][4], fail[maxn], end[maxn];
    int root, cnt;
    int new_node () {
        memset (next[cnt], -1, sizeof next[cnt]);
        end[cnt++] = 0;
        return cnt-1;
    }
    void init () {
        cnt = 0;
        root = new_node ();
    }
    void insert (char *buf) {//字典树插入一个单词
        int len = strlen (buf);
        int now = root;
        for (int i = 0; i < len; i++) {
            int id = buf[i]-'a';
            if (next[now][id] == -1) {
                next[now][id] = new_node ();
            }
            now = next[now][id];
        }
        end[now]++;
    }
    void build () {//构建fail指针
        queue <int> q;
        fail[root] = root;
        for (int i = 0; i < 4; i++) {
            if (next[root][i] == -1) {
                next[root][i] = root;
            }
            else {
                fail[next[root][i]] = root;
                q.push (next[root][i]);
            }
        }
        while (!q.empty ()) {
            int now = q.front (); q.pop ();
            end[now] += end[fail[now]];
            for (int i = 0; i < 4; i++) {
                if (next[now][i] == -1) {
                    next[now][i] = next[fail[now]][i];
                }
                else {
                    fail[next[now][i]] = next[fail[now]][i];
                    q.push (next[now][i]);
                }
            }
        }
    }
    int query (char *buf) {
        int len = strlen (buf);
        for (int i = 0; i < cnt; i++) {
            for (int j = 0; j <= len; j++) dp[i][j] = INF;
        }
        dp[0][0] = 0;
        for (int j = 0; j < len; j++) {
            for (int i = 0; i < cnt; i++) {
                if (end[i]) continue;
                int id = buf[j]-'a';
                int nexti = next[i][id];
                if (!end[nexti]) {//可以走到这个点
                    dp[nexti][j+1] = min (dp[nexti][j+1], dp[i][j]);
                }
                for (int x = 0; x < 4; x++) if (x != id) {
                    nexti = next[i][x];
                    if (!end[nexti]) {
                        dp[nexti][j+1] = min (dp[nexti][j+1], dp[i][j]+1);
                    }
                }
                }
            }
            int ans = INF;
            for (int i = 0; i < cnt; i++) ans = min (ans, dp[i][len]);
            if (ans >= INF)
                ans = -1;
            return ans;
        }
}ac;

int main () {
    int kase = 0;
    while (scanf ("%d", &n) == 1 && n) {
        ac.init ();
        for (int i = 1; i <= n; i++) {
            scanf ("%s", str);
            change (str);
            ac.insert (str);
        }
        ac.build ();
        scanf ("%s", str);
        change (str);
        printf ("Case %d: %d\n", ++kase, ac.query (str));
    }
    return 0;
}
```

`HDU 4511`: 给定n个点的坐标, 每次只能从一个点走到编号比他大的点. 求一条最短的路径, 并且这条路径上不能有给出的子路径. 把这些子路径插进AC自动机里, 然后用$dp[i][j]$表示在图上的i, 字典树上的j的最短路径, 转移方程就是$dp[i][j] = min\left\{dp[p[k] + dis(p,i)|next[k][i] = j\right\}$

```
#define maxn 505
#define INF 1e20

int n, m, k;
int buf[maxn];
double p[maxn][2];

double dis (int i, int j) {
    double x = p[i][0]-p[j][0], y = p[i][1]-p[j][1];
    return sqrt (x*x + y*y);
}

struct trie {
    int next[maxn][55], fail[maxn], end[maxn];
    int root, cnt;
    int new_node () {
        memset (next[cnt], -1, sizeof next[cnt]);
        end[cnt++] = 0;
        return cnt-1;
    }
    void init () {
        cnt = 0;
        root = new_node ();
    }
    void insert (int *buf, int len) {//字典树插入一条路径
        int now = root;
        for (int i = 0; i < len; i++) {
            int id = buf[i];
            if (next[now][id] == -1) {
                next[now][id] = new_node ();
            }
            now = next[now][id];
        }
        end[now]++;
    }
    void build () {//构建fail指针
        queue <int> q;
        fail[root] = root;
        for (int i = 1; i <= n; i++) {
            if (next[root][i] == -1) {
                next[root][i] = root;
            }
            else {
                fail[next[root][i]] = root;
                q.push (next[root][i]);
            }
        }
        while (!q.empty ()) {
            int now = q.front (); q.pop ();
            end[now] += end[fail[now]];
            for (int i = 1; i <= n; i++) {
                if (next[now][i] == -1) {
                    next[now][i] = next[fail[now]][i];
                }
                else {
                    fail[next[now][i]] = next[fail[now]][i];
                    q.push (next[now][i]);
                }
            }
        }
    }
    double dp[55][maxn];//在字典树上i 图上的j
    void query () {
        for (int i = 1; i <= n; i++) {
            for (int j = 0; j < cnt; j++) {
                dp[i][j] = INF;
            }
        }
        dp[1][next[root][1]] = 0;
        for (int i = 1; i < n; i++) {
            for (int j = 0; j < cnt; j++) {
                if (end[j] || dp[i][j] >= INF) continue;
                for (int x = i+1; x <= n; x++) {
                    int nexti = next[j][x];
                    if (end[nexti]) continue;
                    dp[x][nexti] = min (dp[x][nexti], dp[i][j] + dis (i, x));
                }
            }
        }

        double ans = INF;
        for (int i = 0; i < cnt; i++)
            ans = min (ans, dp[n][i]);
        if (ans == INF) {
            printf ("Can not be reached!\n");
        }
        else
            printf ("%.2f\n", ans);
    }
}ac;

int main () {
    //freopen ("in.txt", "r", stdin);
    while (cin >> n >> m && n+m) {
        for (int i = 1; i <= n; i++) {
            cin >> p[i][0] >> p[i][1];
        }
        ac.init ();
        for (int i = 0; i < m; i++) {
            cin >> k;
            for (int j = 0; j < k; j++) {
                cin >> buf[j];
            }
            ac.insert (buf, k);
        }
        ac.build ();
        ac.query ();
    }
    return 0;
}
```

LIS

- O(nlgn)解法

```
//注意INF的值
int h[maxn], dp[maxn];
#define INF 10000000
int LIS (int *a, int n) {
    if (!n)
```

```
        return 0;
    int ans = 0;
    for (int i = 1; i <= n; i++) h[i] = INF;
    for (int i = 1; i <= n; i++) {
        int k = lower_bound (h+1, h+1+n, a[i])-h;
        ans = max (ans, k);
        h[k] = a[i];
    }
    return ans;
}
```

`codeforces 650D`:动态LIS，支持独立的修改 先把询问离线下来按照改变的位置排序，把所有的数字离散化。首先先用树状数组求出每一位为结尾的最长上升前缀f[i]和每一位开始的最长上升后缀g[i]，然后考虑每一个数字改变后的LIS情况，分成两种讨论：

- LIS不包含这个数。这种情况比较简单，考虑是不是每一个LIS都含有这个数字即可，也即对于所有的LIS，某一位必须是这个数。所以直接统计每一个数字在LIS中的位置，然后判断这个位置是否唯一即可。
- LIS包含这个数。对于一个数a[i],显然需要找到一个j和k使得$a_j < a_i < a_k$并且$i < j < k$，直接正反扫两遍用树状数组统计即可。

```
int n, m, cnt;
vector <int> num;
int a[maxn];
struct Q {
    int pos, num, id;
    bool operator < (const Q &a) const {
        return pos < a.pos;
    }
} qu[maxn];
int f[maxn], g[maxn];
int c[maxn];

void add1 (int x, int num) {
    for (int i = x; i < maxn; i += lowbit (i)) get_max (c[i], num);
}
int query1 (int x) {
    int ans = 0;
    for (int i = x; i > 0; i -= lowbit (i)) get_max (ans, c[i]);
    return ans;
}
void add2 (int x, int num) {
    for (int i = x; i > 0; i -= lowbit (i)) get_max (c[i], num);
}
int query2 (int x) {
    int ans = 0;
    for (int i = x; i < maxn; i += lowbit (i)) get_max (ans, c[i]);
    return ans;
}

int LIS;//LIS的长度
```

```
void init () {
    Clear (c, 0);
    for (int i = 1; i <= n; i++) {
        f[i] = query1 (a[i]-1)+1;
        add1 (a[i], f[i]);
    }
    Clear (c, 0);
    for (int i = n; i >= 1; i--) {
        g[i] = query2 (a[i]+1)+1;
        add2 (a[i], g[i]);
    }
    LIS = 0;
    for (int i = 1; i <= n; i++) get_max (LIS, f[i]+g[i]-1);
}

void lisanhua () {
    sort (num.begin (), num.end ());
    num.erase (unique (num.begin (), num.end ()), num.end ());
    cnt = num.size ();
    for (int i = 1; i <= n; i++)
        a[i] = lower_bound (num.begin (), num.end (), a[i])-num.begin ()+1;
    for (int i = 1; i <= m; i++)
        qu[i].num = lower_bound (num.begin (), num.end (), qu[i].num)-num.begin ()+1;
}

int ans1[maxn];//不包含每一个数的LIS
int tmp[maxn];
void solve1 () {
    Clear (tmp, 0);
    for (int i = 1; i <= n; i++) if (f[i]+g[i]-1 == LIS) {
        tmp[f[i]]++;
    }
    for (int i = 1; i <= n; i++) {
        if (f[i]+g[i]-1 == LIS && tmp[f[i]] == 1) ans1[i] = LIS-1;
        else ans1[i] = LIS;
    }
}

//包含每一个改变值的LIS
int ans[maxn];//最后的答案
int l[maxn], r[maxn];//左边比他小的最长 右边比他大的最长
void solve2 () {
    Clear (c, 0);
    int pos = 1;
    for (int i = 1; i <= m; i++) {
        while (pos < qu[i].pos) {
            add1 (a[pos], f[pos]);
            pos++;
        }
        l[i] = query1 (qu[i].num-1);
    }
    Clear (c, 0);
    pos = n;
    for (int i = m; i >= 1; i--) {
        while (pos > qu[i].pos) {
            add2 (a[pos], g[pos]);
            pos--;
        }
        r[i] = query2 (qu[i].num+1);
    }
    for (int i = 1; i <= m; i++) {
        ans[qu[i].id] = ans1[qu[i].pos];
        get_max (ans[qu[i].id], l[i]+r[i]+1);
    }
}

int main () {
    scanf ("%d%d", &n, &m);
    num.clear ();
    for (int i = 1; i <= n; i++) {
        scan (a[i]);
        num.pb (a[i]);
    }
    for (int i = 1; i <= m; i++) {
        scanf ("%d%d", &qu[i].pos, &qu[i].num);
        num.pb (qu[i].num);
        qu[i].id = i;
    }
    sort (qu+1, qu+1+m);
    lisanhua ();
    init ();
    solve1 ();
    solve2 ();
    for (int i = 1; i <= m; i++) {
        printf ("%d\n", ans[i]);
    }
    return 0;
}
```

数位DP

`HDU 2089` : [l,r]没有连续62或者4的数字个数。

```
long long n, m;
int bit[11], l;
long long dp[11][2];

long long dfs (int pos, bool f1, bool f2) {
//当前的位数 前一个数是不是6 是不是可以取到9
```

```
    if (pos == 0)
        return 1;
    if (f2 && dp[pos][f1] != -1) {
        return dp[pos][f1];
    }
    int Max = (f2 ? 9 : bit[pos]);
    long long ans = 0;
    for (int i = 0; i <= Max; i++) {
        if (i == 4 || (f1 && i == 2))
            continue;
        ans += dfs (pos-1, i==6, f2 || i<Max);
    }
    if (f2)
        dp[pos][f1] = ans;
    return ans;
}

long long f (long long num) {
    l = 0;
    while (num) {
        bit[++l] = num%10;
        num /= 10;
    }
    return dfs (l, 0, 0);
}

int main () {
    memset (dp, -1, sizeof dp);
    while (cin >> n >> m && n+m) {
        cout << f (m)-f (n-1) << endl;
    }
    return 0;
}
```

`hihocoder 1259`: 定义g(x)为f(1)到f(n)中模k余n的个数,求出所有g(x)的异或和. f(2x)=3f(x),f(2x+1)=f(2x)+1.然后发现f(1)到f(n)在三进制下等价于二进制下的1到n,然后就可以用数位DP搞了.这个思想下的转移方程dp(d,m) = dp(d-1, (m-x)%k).

```
long long dp[68][68686];
long long n, ans[68686];
int mod;
int bit[68], l;
#define pow Pow
long long pow[68];

void solve (long long num) {
    long long cur = 1;
    l = 0;
    while (num) {
        bit[++l] = (num&1);
        num >>= 1;
    }
    pow[0] = 1;
    for (int i = 1; i <= l; i++) pow[i] = pow[i-1]*3%mod;
    for (int i = 1; i <= l; i++, cur *= 3, cur %= mod) {
        for (int j = 0; j < mod; j++) {
            dp[i][(j+cur)%mod] += dp[i-1][j];//1
            dp[i][j] += dp[i-1][j];//0
        }
    }
    memset (ans, 0, sizeof ans);
    cur = 0;
    for (int i = l; i >= 1; i--) {
        if (bit[i]) {
            for (int j = 0; j < mod; j++) {
                ans[(j+cur)%mod] += dp[i-1][j];
            }
            cur += pow[i-1]%mod;
            cur %= mod;
        }
    }
    ans[cur]++;//最终的数字
    ans[0]--;//排除数字0
    long long gg = 0;
    for (int i = 0; i < mod; i++) {
        gg ^= ans[i];
    }
    cout << gg << endl;
    return ;
}

int main () {
    //freopen ("in.txt", "r", stdin);
    int t;
    scanf ("%d", &t);
    while (t--) {
        memset (dp, 0, sizeof dp);
        dp[0][0] = 1;
        scanf ("%lld%d", &n, &mod);
        solve (n);
    }
    return 0;
}
```

概率(期望)DP

`POJ 2096` : 有s个系统,有n种bug,bug的数量不限,一位程序员每天可以发现一个bug.求发现n种bug存在s个系统中并且每个系统都要被发现bug的平均天数(期望)

~~~~
```
在这里有s,n两个变量，假设dp[i][j]表示已经发现i种bug存在j个系统到目标所需要的天数,显然dp[n][s]=0;
从dp[i][j]经过一天后可能得到以下四种情况:
dp[i][j]->dp[i+1][j+1];//在一个新的系统里面发现一个新的bug
dp[i][j]->dp[i+1][j];//在原来已发现过bug的系统里发现一个新的bug
dp[i][j]->dp[i][j+1];//在一个新的系统里面发现一个已被发现过的bug
dp[i][j]->dp[i][j];//在已发现过bug的系统发现已发现过的bug
四种到达的概率分别是:
p1=(n-i)*(s-j)/(n*s);
p2=(n-i)*j/(n*s);
p3=i*(s-j)/(n*s);
p4=i*j/(n*s);
然后根据E(aA+bB+cC+dD+...)=aEA+bEB+....;//a,b,c,d...表示概率,A,B,C...表示状态
所以dp[i][j]=p1*dp[i+1][j+1]+p2*dp[i+1][j]+p3*dp[i][j+1]+p4*dp[i][j]+1;//dp[i][j]表示的
就是到达状态i,j的期望
=>dp[i][j]=(p1*dp[i+1][j+1]+p2*dp[i+1][j]+p3*dp[i][j+1]+1)/(1-p4);
```
~~~~

斜率优化

一般先确定一个j比k优的条件，把条件推成斜率式子。

`HDU 3840`： 给出一个数字集合S, 要求m个子集, 使得子集并等于S, 并且每一个子集的花费和最小. 一个集合的花费等于最大元素减去最小元素的平方. 显然贪心的想一下是最后肯定是数列排完序之后划分一下. 然后$cost_{i,j} = (a_j - a_i)^2$, 转移方程就是$f_i = min\left\{f_j + (a_i - a_{j+1})^2 \middle| j < i\right\}$, 于是对于每一个下标, 直接枚举前面要n的复杂度, 可以确定j比k优的条件是$f_j + (a_i - a_{j+1})^2 \le f_k + (a_i - a_{k+1})^2$。整理下就是$\frac{f_j + a_{j+1}^2 - f_k - a_{k+1}^2}{2a_{j+1} - 2a_{k+1}} \le a_i$

```
#define maxn 10005
int dp[maxn][5005];
int n, m, que[maxn];
int a[maxn];

int up (int i, int j, int k) {
    return dp[i][k]+a[i+1]*a[i+1] - (dp[j][k]+a[j+1]*a[j+1]);
}

int down (int i, int j, int k) {
    return 2*(a[i+1]-a[j+1]);
}

int main () {
    int kase = 0, t;
    scanf ("%d", &t);
    while (t--) {
        printf ("Case %d: ", ++kase);
        scanf ("%d%d", &n, &m);
        for (int i = 1; i <= n; i++) scanf ("%d", &a[i]);
        if (m >= n) {
```

```
            printf ("0\n");
            continue;
        }
        m--;
        sort (a+1, a+1+n);
        for (int i = 1; i <= n; i++) dp[i][0] = (a[i]-a[1])*(a[i]-a[1]);

        for (int k = 1; k <= m; k++) {
            int L = 0, R = 0;
            que[R++] = 0; //有时候需要DP值的初始化
            for (int i = 1; i <= n; i++) {
                while (L+1 < R && up (que[L+1], que[L], k-1) <= a[i]*down (que[L+1],
que[L], k-1)) //这里的等号不要忘记!
                    L++;
                int j = que[L];
                dp[i][k] = dp[j][k-1] + (a[i]-a[j+1])*(a[i]-a[j+1]);
                while (L+1 < R && up (i, que[R-1], k-1)*down (que[R-1], que[R-2], k-
1) <=
                       up (que[R-1], que[R-2], k-1)*down (i, que[R-1], k-1))
                    R--;
                que[R++] = i;
            }
        }
        printf ("%d\n", dp[n][m]);
    }
    return 0;
}
```

背包

- 多重背包

每种物品按照数量/价值拆开成$logn$个物品，转化成普通的01背包问题。

```
//cnt[i]第i个物品的数量 m[i]第i个物品的重量 Max背包的容量
for (int i = 1; i <= n; i++) if (cnt[i]) {
    for (int bit = 1; bit <= cnt[i]; bit <<= 1) {
        for (int j = Max; j >= 0; j--) if (j-bit*m[i] >= 0 && dp[j-bit*m[i]]) {
            dp[j] = 1;
        }
        cnt[i] -= bit;
    }
    if (cnt[i]) {
        for (int j = Max; j >= 0; j--) if (j-cnt[i]*m[i] >= 0 && dp[j-cnt[i]*m[i]]) {
            dp[j] = 1;
        }
    }
}
```

搜索

dfs

剪枝

`HDU 1455`: 求n个木棒最多拼成多少个长度相等的木棒。

```
#define maxn 66

int a[maxn], n;
int sum, len, cnt;
bool vis[maxn];

bool cmp (const int &a, const int &b) {
    return a > b;
}

bool dfs (int pos, int cur, int num) {
//当前枚举棒子的下标 当前这段棒子的长度 已经拼了num段长为len的棒子
    if (num == cnt) {
        return 1;
    }
    if (pos > n) {
        return 0;
    }
    if (cur == len) {
        if (dfs (0, 0, num+1))
            return 1;
    }
    for (int i = pos; i < n; i++) {
        if (!vis[i] && cur+a[i] <= len) {
            vis[i] = 1;//使用这个木棒
            if (dfs (i+1, cur+a[i], num))
                return 1;
            vis[i] = 0;//回溯时清除标记
            if (cur+a[i] == len)
            //剪枝1：后面几个小棒子组成的a[i]长度也没有意义
                return 0;
            if (a[i] == a[i+1])
            //剪枝2：后面一个棒子等于当前可以跳过
                i++;
            if (cur == 0)
            //剪枝3：因为cur=0当前木棍必须用到，但是搜索结果不为真
            //可以判断之前的选取无效
                return 0;
        }
    }
    return 0;
}
```

```
int main () {
    while (scanf ("%d", &n) == 1 && n) {
        sum = 0;
        for (int i = 0; i < n; i++) {
            scanf ("%d", &a[i]);
            sum += a[i];
        }
        a[n] = 0;
        sort (a, a+n, cmp);
        for (int i = a[0]; i <= sum; i++) {//贪心枚举
            if (sum%i != 0)
                continue;
            len = i;
            cnt = sum/i;//需要将棒子拼成cnt段相同的，每段长len
            memset (vis, 0, sizeof vis);//清空访问标记
            if (dfs (0, 0, 0)) {//如果找到一种可行解
                printf ("%d\n", i);
                break;
            }
        }
    }
    return 0;
}
```

随机背包(大背包)

`HDU 5887`：给出n个物品，体积和价值都是1e9以内的随机数。求获取的最大价值。

```
int n, vLimit, v[100], w[100];
long double rate[100];
long long wBest;

void swap(int i, int j) {
    int tmp;
    tmp = v[i];
    v[i] = v[j];
    v[j] = tmp;
    tmp  = w[i];
    w[i] = w[j];
    w[j] = tmp;
    double tmp2;
    tmp2 = rate[i];
    rate[i] = rate[j];
    rate[j] = tmp2;
}


void sort() {
    for (int i = 0; i < n; ++ i) {
```

```cpp
        for (int j = i + 1; j < n; ++ j) {
            if (rate[i] < rate[j]) swap(i, j);
        }
    }
}

bool can_cut(int k, int _vCurrent, long long _wCurrent) {
    long double vCurrent = _vCurrent;
    long double wCurrent = _wCurrent;
    for (int i = k; i < n && vCurrent < vLimit; ++ i) {
        if (vCurrent + v[i] < vLimit) {
            vCurrent += v[i];
            wCurrent += w[i];
        } else {
            wCurrent += rate[i] * (vLimit - vCurrent);
            vCurrent = vLimit;
        }
    }
    return wCurrent < wBest;
}

void search(int k, int vSum, long long wSum) {
    if (wSum > wBest) {
        wBest = wSum;
    }
    if (k < n && !can_cut(k, vSum, wSum)) {
        if ((long long)vSum + v[k] <= vLimit) {
            search(k + 1, vSum + v[k], wSum + w[k]);
        }
        search(k + 1, vSum, wSum);
    }
}

int main() {
    while (scanf("%d%d", &n, &vLimit) != EOF) {
        for (int i = 0; i < n; ++ i) {
            scanf("%d%d", &v[i], &w[i]);
            rate[i] = (long double)w[i] / (long double)v[i];
        }
        sort();
        wBest = 0;
        search(0, 0, 0);
        std::cout << wBest << std::endl;
    }
}
```

其他

贪心

`HDU 4912`:树上选择最多的链使得两两不交. 按照链的LCA排序，自底向上贪心选择。

```
#define maxn 100005
#define maxm maxn*2

int n, m;
struct node {
    int v, next;
}edge[maxm];
int cnt, head[maxn];
int deep[maxn];
struct Q {
    int u, v;
    int lca;
    int next;
    bool operator < (const Q &a) const {
        return deep[lca] > deep[a.lca];
    }
}qu[maxm];
int cnt2, head2[maxn];
int p[maxn], fa[maxn];
bool vis[maxn];

#define find Find
int find (int x) {
    return p[x] == x ? p[x] : p[x] = find (p[x]);
}

void add_edge (int u, int v) {
    edge[cnt].v = v, edge[cnt].next = head[u], head[u] = cnt++;
}

void add_query (int u, int v) {
    qu[cnt2].u = u, qu[cnt2].v = v, qu[cnt2].next = head2[u], head2[u] = cnt2++;
}

void LCA (int u, int father, int d) {
    deep[u] = d;
    p[u] = u;
    fa[u] = father;
    vis[u] = 1;
    for (int i = head[u]; i != -1; i = edge[i].next) {
        int v = edge[i].v;
        if (vis[v]) continue;
        LCA (v, u, d+1);
        p[v] = u;
    }
    for (int i = head2[u]; i != -1; i = qu[i].next) {
        int v = qu[i].v;
        if (vis[v]) {
            qu[i^1].lca = qu[i].lca = find (v);
        }
    }
}

void del (int u) {
    vis[u] = 1;
    for (int i = head[u]; i != -1; i = edge[i].next) {
        int v = edge[i].v;
        if (vis[v] || v == fa[u]) continue;
        del (v);
    }
}

void solve () {
    int ans = 0;
    for (int i = 0; i < cnt2; i++) {
        if (vis[qu[i].u] || vis[qu[i].v]) continue;
        ans++;
        del (qu[i].lca);
    }
    printf ("%d\n", ans);
}

int main () {
    while (scanf ("%d%d", &n, &m) == 2) {
        cnt = cnt2 = 0;
        for (int i = 1; i <= n; i++) head[i] = head2[i] = -1, vis[i] = 0;
        for (int i = 1; i < n; i++) {
            int u, v;
            scanf ("%d%d", &u, &v);
            add_edge (u, v);
            add_edge (v, u);
        }
        for (int i = 1; i <= m; i++) {
            int u, v;
            scanf ("%d%d", &u, &v);
            add_query (u, v);
            add_query (v, u);
        }
        LCA (1, 1, 0);
        cnt2 >>= 1;
        for (int i = 0; i < cnt2; i++) {
            qu[i] = qu[i<<1];
        }
        sort (qu, qu+cnt2);
        memset (vis, 0, sizeof vis);
        solve ();
    }
    return 0;
}
```

构造

`BZOJ 4319`:构造一个串满足给定的后缀数组($sa[]$)。

从最小的后缀开始构造，如果$rank[sa[i-1]+1] > rank[sa[i]+1]$，那么就表示排名第$i$的后缀字母必须比$i-1$的大。

前后缀

最大子序列/矩阵和

最大子段和：枚举末尾j，维护到j之前的前缀和最小值即可。 最大子矩阵和：枚举i, j,用这两行卡着子矩阵的上下届，然后就变成了一维的最大子序列和问题。

`opencup 6298F`:交换两个数字以后的最大子段和和交换方案。

交换必然是一个较大的和较小的交换，并且是段内和段外交换（特殊情况排除），分两者的左右关系讨论。如果较大的在右边，假设交换的是(i, j)，那么被选择的子段必然是以$i-1$为结尾的最大子段和$i+1$往右的某些连续数字，枚举右边的边界即可，另一种情况同理。

```
int n; long long a[maxn];
long long l[maxn], r[maxn], lsum[maxn], rsum[maxn]; pli maxr[maxn], maxl[maxn];

void solve1 (long long &Maxsum, pii &ans) {
    long long Max, pos;
    Maxsum = -INF;
    Max = -lsum[1]+l[0], pos = 1;
    for (int i = 2; i < n; i++) {
        if (-lsum[i]+l[i-1] > Max) {
            Max = -lsum[i]+l[i-1];
            pos = i;
        }
        long long tmp = lsum[i]+Max+maxr[i+1].fi;
        if (tmp > Maxsum) {
            Maxsum = tmp;
            ans = make_pair (pos, maxr[i+1].se);
        }
    }
}

void solve2 (long long &Maxsum, pii &ans) {
    long long Max, pos;
    Max = -rsum[n], pos = n;
    for (int i = n-1; i > 1; i--) {
        if (-rsum[i]+r[i+1] > Max) {
            Max = -rsum[i]+r[i+1];
            pos = i;
        }
        long long tmp = rsum[i]+Max+maxl[i-1].fi;
```

```
            if (tmp > Maxsum) {
                Maxsum = tmp;
                ans = make_pair (maxl[i-1].se, pos);
            }
        }
}

int main () {
    while (cin >> n) {
        lsum[0] = 0; l[0] = 0;
        for (int i = 1; i <= n; i++) {
            cin >> a[i];
            lsum[i] = lsum[i-1]+a[i];
            l[i] = max (l[i-1]+a[i], max (a[i], 0LL));
            maxl[i] = (i == 1 ? make_pair (a[i], 1) : max (maxl[i-1], make_pair
(a[i], i)));
        }
        if (n == 2) {
            cout << max (max (a[1], a[2]), a[1]+a[2]) << endl << "1 2" << endl;
            continue;
        }
        maxr[n] = make_pair (a[n], n), r[n] = (a[n] > 0 ? a[n] : 0);
        rsum[n] = a[n];
        for (int i = n-1; i >= 1; i--) {
            maxr[i] = max (maxr[i+1], make_pair (a[i], i));
            r[i] = max (r[i+1]+a[i], max (0LL, a[i]));
            rsum[i] = rsum[i+1]+a[i];
        }
        long long Maxsum; pii ans;
        solve1 (Maxsum, ans);
        solve2 (Maxsum, ans);
        if (lsum[n] > Maxsum) {
            Maxsum = lsum[n], ans = make_pair (1, 2);
        }
        cout << Maxsum << endl << ans.fi << " " << ans.se << endl;
    }
    return 0;
}
```

分治&&CDQ分治

三维偏序

`HDU 5618` : 每个点求出三个坐标都小于等于他的点的个数.

一维排序，然后分治区间[l,r]，递归处理[l,mid]，然后计算[l,mid]对于[mid+1,r]的贡献。用一个副本copy两段区间都按照第二维排序，然后用一个双指针扫第二维，第三维用树状数组维护。计算完贡献之后递归处理[mid+1,r]。

```
struct node {
    int id;
```

```cpp
    int x, y, z;
    bool operator == (const node &a) const {
        return x == a.x && y == a.y && z == a.z;
    }
}qu[maxn], tmp[maxn];
int ans[maxn];
int c[maxn];
int n, Max;

bool cmp1 (const node &a, const node &b) {
    if (a.z != b.z)
        return a.z < b.z;
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}

bool cmp2 (const node &a, const node &b) {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}

int lowbit (int x) {
    return x&(-x);
}

void add (int x) {
    for (int i = x; i < maxn; i += lowbit (i))
        c[i]++;
}

int sum (int x) {
    int ans = 0;
    for (int i = x; i > 0; i -= lowbit (i))
        ans += c[i];
    return ans;
}

void clear (int x) {
    for (int i = x; i < maxn; i += lowbit (i))
        c[i] = 0;
}

void solve (int l, int r) {
    if (l == r)
        return ;
    int mid = (l+r)>>1;
    solve (l, mid);
    for (int i = l; i <= r; i++) tmp[i] = qu[i];
    sort (tmp+l, tmp+mid+1, cmp2);
    sort (tmp+mid+1, tmp+r+1, cmp2);
    int L = l;
    for (int i = mid+1; i <= r; i++) {
        while (L <= mid && tmp[L].x <= tmp[i].x) {
            add (tmp[L].y);
            L++;
        }
        ans[tmp[i].id] += sum (tmp[i].y);
    }
    for (int i = l; i <= mid; i++) {
        clear (tmp[i].y);
    }
    solve (mid+1, r);
}

int main () {
    int t;
    scanf ("%d", &t);
    while (t--) {
        scanf ("%d", &n);
        memset (c, 0, sizeof c);
        memset (ans, 0, sizeof ans);
        for (int i = 1; i <= n; i++) {
            scanf ("%d%d%d", &qu[i].x, &qu[i].y, &qu[i].z);
            qu[i].id = i;
        }
        sort (qu+1, qu+1+n, cmp1);
        solve (1, n);
        for (int i = n-1; i >= 1; i--) {
            if (qu[i] == qu[i+1])
                ans[qu[i].id] = ans[qu[i+1].id];
        }
        for (int i = 1; i <= n; i++) {
            printf ("%d\n", ans[i]);
        }
    }
    return 0;
}
```

`BZOJ 3295`:动态逆序对。给出一个1-n的排列，每次询问删除某一个值，输出逆序对数。

给所有的数增加一个时间序，相当于求三维偏序。

```
int n, m, a[maxn];
int t[maxn], f[maxn], g[maxn];
int qu[maxn];
long long ans[maxn];

int tmp[maxn], c[maxn];
bool cmp1 (const int &i, const int &j) {
    return i < j;
}

bool cmp2 (const int &i, const int &j) {
    return i > j;
}

void add (int num) {
    for (int i = num; i < maxn; i += lowbit (i)) c[i]++;
}

long long query (int num) {
    long long ans = 0;
    for (int i = num; i > 0; i -= lowbit (i)) ans += c[i];
    return ans;
}

void del (int num) {
    for (int i = num; i < maxn; i += lowbit (i)) c[i] = 0;
}

void cdq1 (int l, int r) {
    int mid = (l+r)>>1;
    if (l == r) return ;
    cdq1 (l, mid);
    for (int i = l; i <= r; i++) tmp[i] = a[i];
    sort (tmp+l, tmp+mid+1, cmp2);
    sort (tmp+mid+1, tmp+r+1, cmp2);
    int pos = l;
    for (int i = mid+1; i <= r; i++) {
        while (pos <= mid && tmp[pos] > tmp[i]) {
            add (t[tmp[pos]]);
            pos++;
        }
        f[tmp[i]] += query (t[tmp[i]]);
    }
    for (int i = l; i <= mid; i++) {
        del (t[tmp[i]]);
    }
    cdq1 (mid+1, r);
}

void cdq2 (int l, int r) {
    int mid = (l+r)>>1;
    if (l == r) return ;
    cdq2 (mid+1, r);
    for (int i = l; i <= r; i++) tmp[i] = a[i];
    sort (tmp+l, tmp+mid+1, cmp1);
    sort (tmp+mid+1, tmp+r+1, cmp1);
    int pos = mid+1;
    for (int i = l; i <= mid; i++) {
        while (pos <= r && tmp[pos] < tmp[i]) {
            add (t[tmp[pos]]);
            pos++;
        }
```

```
            g[tmp[i]] += query (t[tmp[i]]);
        }
        for (int i = mid+1; i <= r; i++) {
            del (t[tmp[i]]);
        }
        cdq2 (l, mid);
    }

    int main () {
        while (scanf ("%d%d", &n, &m) == 2) {
            Clear (c, 0);
            for (int i = 1; i <= n; i++) {
                f[i] = 0, g[i] = 0;
                scan (a[i]);
                t[qu[i]] = 0;
            }
            for (int i = 1; i <= m; i++) {
                scan (qu[i]);
                t[qu[i]] = (n-m)+(m-i)+1;
            }
            int cnt = 0;
            for (int i = 1; i <= n; i++) if (t[i] == 0) t[i] = ++cnt;

            cdq1 (1, n);
            cdq2 (1, n);
            long long sum = 0;
            for (int i = 1; i <= n; i++) if (t[i] <= n-m) {
                sum += f[i]+g[i];
            }
            for (int i = m; i >= 1; i--) {
                ans[i] = (sum += f[qu[i]]+g[qu[i]]);
            }
            for (int i = 1; i <= m; i++) printf ("%lld\n", ans[i]);
        }
        return 0;
    }
```

####求解卷积

(见FFT&&NTT)

拓展

java

- java排序：

```
Arrays.sort (数组名,起始下标,终止下标);
```

- 类型转化：

```
String a;
double x = Double.PraseDouble(a); //string转double
```

- 字符串：

```
String a = cin.next();//读取
char ch = a.charAt(pos);//取下标
```

- a+b大数版

```
 import java.math.BigInteger;
 import java.util.Scanner;

 public class Main {
     void solve () {
         BigInteger a, b, c;
         Scanner cin = new Scanner(System.in);
         int t = cin.nextInt ();
         for (int i = 1; i <= t; i++) {
             System.out.println ("Case " + i + ":");
             a = cin.nextBigInteger ();
             b = cin.nextBigInteger ();
             System.out.println (a + " + " + b + " = " + a.add (b));
             if (i != t) System.out.println ();
         }
     }
     public static void main (String[] args) {
         Main work = new Main();
         work.solve ();
     }
 }
```

- 高精度小数，要去掉末尾的后导0.

```
//无法整除时需要保留小数位数
Bigdecimal a, b;
a = a.divide (b, 100, BigDecimal.ROUND_CEILING); //被除数 位数 小数截断方式
setScale (1, BigDecimal.ROUND_HALF_UP); //保留一位小数
```

```
 import java.math.*;
 import java.util.*;

 public class Main {
     void solve () {
         Scanner cin = new Scanner(System.in);
         BigDecimal a = BigDecimal.valueOf (0);
```

```
        BigDecimal b = BigDecimal.valueOf (0);
        while (cin.hasNext ()) {
            a = cin.nextBigDecimal ();
            b = cin.nextBigDecimal ();
            System.out.println (a.add (b).stripTrailingZeros().toPlainString());
        }
    }
    public static void main (String[] args) {
        Main work = new Main();
        work.solve ();
    }
}
```

`BZOJ 2852` :给定两个小数a,b,求一个最小的k使得[ak,bk]存在整数。

类似于辗转相除，设$ak \le t \le bk$，化成$t/b \le k \le t/a$，不等式同时减掉一个数字不影响正确性，这样区间[a,b]就变成了处理区间[1/a,1/b].

```
public class Main
{
    BigDecimal dfs (BigDecimal a, BigDecimal b)
    {
        BigDecimal c = a;

        c = c.setScale(0, BigDecimal.ROUND_DOWN);

        a = a.subtract(c);
        b = b.subtract(c);
        if (b.compareTo(BigDecimal.ONE) >= 1) return BigDecimal.ONE;
        b = BigDecimal.ONE.divide(b, 1000, BigDecimal.ROUND_HALF_UP);
        if (BigDecimal.ZERO.compareTo(a.setScale(1000, BigDecimal.ROUND_HALF_UP)) ==
0)
        {
            return b.setScale(0, BigDecimal.ROUND_UP);

        }
        a = BigDecimal.ONE.divide(a, 1000, BigDecimal.ROUND_HALF_UP);
        BigDecimal ans = dfs (b, a);
        b = b.multiply(ans);
        return b.setScale(0, BigDecimal.ROUND_UP);

    }
    void solve ()
    {
        int T;
        Scanner cin = new Scanner(System.in);
        T = cin.nextInt();
        BigDecimal a, b;
        for (int i = 1; i <= T; i++)
        {
            a = cin.nextBigDecimal();
            b = cin.nextBigDecimal();
            BigDecimal ans = dfs(a, b);
            ans = ans.setScale(0, BigDecimal.ROUND_DOWN);
            //注释部分是解决左闭右开 HDU6238•
            /*BigDecimal bb = ans.multiply(b), aa = ans.multiply(a);
            if (bb.compareTo(bb.setScale(0, BigDecimal.ROUND_DOWN)) == 0 &&
bb.subtract(aa).compareTo(BigDecimal.ONE) < 0) {
                                ans = ans.add(BigDecimal.ONE);            }*/
            System.out.println(ans.setScale(0, BigDecimal.ROUND_DOWN).toString());

        }
    }   public static void main (String[] args)
    {
        Main work = new Main();
        work.solve ();

    }
}
```

输入输出外挂

黑科技文件读入

```
namespace fastIO {
    #define BUF_SIZE 100000
    //fread -> read
    bool IOerror = 0;
    inline char nc() {
        static char buf[BUF_SIZE], *p1 = buf + BUF_SIZE, *pend = buf + BUF_SIZE;
        if(p1 == pend) {
            p1 = buf;
            pend = buf + fread(buf, 1, BUF_SIZE, stdin);
            if(pend == p1) {
                IOerror = 1;
                return -1;
            }
        }
        return *p1++;
    }
    inline bool blank(char ch) {
        return ch == ' ' || ch == '\n' || ch == '\r' || ch == '\t';
    }
    inline void read(int &x) {
        char ch;
        while(blank(ch = nc()));
        if(IOerror)
            return;
        for(x = ch - '0'; (ch = nc()) >= '0' && ch <= '9'; x = x * 10 + ch - '0');
    }
    #undef BUF_SIZE
};

while (fastIO::read (n), !fastIO::IOerror) {}
```

```
#define readBuf() { \
    if (++ bufb == bufe) \
        bufe = (bufb = buf) + fread(buf, 1, sizeof(buf), stdin); \
}
#define readInt(_y_) { \
    register int _s_(0); \
    do { \
        readBuf(); \
    } while (!isdigit(*bufb)); \
    do { \
        _s_ = _s_ * 10 + *bufb - 48; \
        readBuf(); \
    } while (isdigit(*bufb)); \
    _y_ = _s_; \
}
#define reads(_y_) { \
    CH = -1; \
    do { \
        readBuf(); \
    } while (*bufb < '!'); \
    do { \
        _y_[++CH] = *bufb; \
        readBuf(); \
    } while (*bufb > '!'); \
    _y_[CH+1] = 0; \
}
```

求某天是星期几

基姆拉尔森计算公式:

```
int CaculateWeekDay(int y,int m,int d){
//星期数是0-6
    int a;
    // 1月2月当作前一年的13,14月
    if (m==1||m==2){
        m += 12;
        y--;
    }
    //判断是否在1752年9月3号之前
    if(y<1752||(y==1752&&m<9)||(y==1752&&m==9&&d<3)){
        a =  (d+2*m+3*(m+1)/5+y+y/4+5)%7;
    }
    else {
        a = (d+2*m+3*(m+1)/5+y+y/4-y/100+y/400)%7;
    }
    return a;
}
```

扩栈

```
#pragma comment(linker, "/STACK:1024000000,1024000000")
```

头文件组

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <queue>
#include <cmath>
#include <algorithm>
#include <stack>
#include <map>
#define Clear(x,y) memset (x,y,sizeof(x))
#define Close() ios::sync_with_stdio(0)
#define Open() freopen ("more.in", "r", stdin)
#define get_min(a,b) a = min (a, b)
#define get_max(a,b) a = max (a, b);
#define y0 yzz
#define y1 yzzz
#define fi first
#define se second
#define pii pair<int, int>
#define pli pair<long long, int>
#define pll pair<long long, long long>
#define pdi pair<double, int>
#define pdd pair<double, double>
#define pb push_back
#define pl c<<1
#define pr (c<<1)|1
#define lson l,mid,pl
#define rson mid+1,r,pr
typedef unsigned long long ull;
template <class T> inline T lowbit (T x) {return x&(-x);}
template <class T> inline T sqr (T x) {return x*x;}
template <class T>
inline bool scan (T &ret) {
    char c;
    int sgn;
    if (c = getchar(), c == EOF) return 0; //EOF
    while (c != '-' && (c < '0' || c > '9') ) c = getchar();
    sgn = (c == '-') ? -1 : 1;
    ret = (c == '-') ? 0 : (c - '0');
    while (c = getchar(), c >= '0' && c <= '9') ret = ret * 10 + (c - '0');
    ret *= sgn;
    return 1;
}
std::ostream& operator<<(std::ostream& os, __int128 T) {
    if (T<0) os<<"-";if (T>=10 ) os<<T/10;if (T<=-10) os<< -(T/10);
    return os<<( (int) (T%10) >0 ? (int) (T%10) : -(int) (T%10) );
}
const double pi = 3.14159265358979323846264338327950288L;
using namespace std;
#define mod 1000000007
#define INF 1e9
#define maxn 1000005
#define maxm 1000005
//-----------------morejarphone--------------------//
```

vimrc简单配置

```
set tabstop=4
set autoindent
set shiftwidth=4
set cindent
set nu
set hlsearch
set backspace=2
set mouse=a
```