# Standard Code Library
# of
# HUST Affiliated Kindergarten

Verson 1.2

Edited By

sddyzjh

DragoonKiller

dreactor

*Huazhong University of Science and Technology*

# 目录

# 计算几何

## 几何通用

```
/// 计算几何专用. 按需选用.

db eps = 1e-12; // 线性误差范围; long double : 1e-16;
db eps2 = 1e-6; // 平方级误差范围; long double: 1e-8;
bool eq(db a, db b) { return abs(a-b) < eps; }

// ========================= 点和向量 ==========================
struct point;
struct point
{
    db x, y;
    point():x(0),y(0) { }
    point(db a,db b):x(a),y(b) { }
    point(point const& f):x(f.x),y(f.y) { }
    point operator=(point const& f) { x=f.x; y=f.y; return *this; }

    point operator+(point const& b) const { return point(x + b.x, y + b.y); }
    point operator-(point const& b) const { return point(x - b.x, y - b.y); }
    point operator()(point const& b) const { return b - *this; } // 从本顶点出发,指向另一个点的向量.

    db len2() const { return x*x+y*y; } // 模的平方.
    db len() const { return sqrt(len2()); } // 向量的模.
    point norm() const { db l = len(); return point(x/l, y/l); } // 标准化.

    // 把向量旋转f个弧度.
    point rot(double const& f) const
    { return point(x*cos(f) - y*sin(f), x*sin(f) + y*cos(f)); }

    // 极角, +x轴为0, 弧度制, (- , ].
    db pangle() const { if(y >= 0) return acos(x / len()); else return -acos(x / len()); }

    void out() const { printf("(%.2f, %.2f)", x, y); } // 输出.
};

// 数乘.
point operator*(point const& a, db const& b) { return point(a.x * b, a.y * b); }
point operator*(db const& b, point const& a) { return point(a.x * b, a.y * b); }

// 叉积.
db operator*(point const& a, point const& b) { return a.x * b.y - a.y * b.x;
```

```
    }
// 点积.
db operator&(point const& a, point const& b) { return a.x * b.x + a.y * b.y; }

bool operator==(point const& a, point const& b) { return eq(a.x, b.x) && eq(a.y, b.y); }

// 判断本向量在另一个向量的顺时针方向. 注意选用eps或0.
bool operator>>(point const& a, point const& b) { return a*b > eps; }
// 判断本向量在另一个向量的顺时针方向或同向. 注意选用eps或0.
bool operator>>=(point const& a, point const& b) { return a*b > -eps; }

// ============================== 线段 ==============================
struct segment
{
    point from,to;
    segment(point const& a = point(), point const& b = point()) : from(a), to(b) { }

    point dir() const { return to - from; } // 方向向量,未标准化.

    db len() const { return dir().len(); } // 长度.

    // 点在线段上.
    bool overlap(point const& v) const
    { return eq(from(to).len(), v(from).len() + v(to).len()); }

    point projection(point const& p) const // 点到直线上的投影.
    {
        db h = abs(dir() * from(p)) / len();
        db r = sqrt(from(p).len2() - h*h);
        if(eq(r, 0)) return from;
        if((from(to) & from(p)) < 0) return from(to).norm() * (-r);
        else return from(to).norm() * r;
    }

    point nearest(point const& p) const // 点到线段的最近点.
    {
        point g = projection(p);
        if(overlap(g)) return g;
        if(g(from).len() < g(to).len()) return from;
        return to;
    }
};

bool operator/(segment const& a, segment const& b) // 平行 (零向量平行于任意向量).
{
    return eq(a.dir() * b.dir(), 0);
}
```

```

// 相交. 不计线段端点则删掉 eq(..., 0) 的所有判断.
bool operator*(segment const& A, segment const& B)
{
    point dia = A.from(A.to);
    point dib = B.from(B.to);
    db a = dia * A.from(B.from);
    db b = dia * A.from(B.to);
    db c = dib * B.from(A.from);
    db d = dib * B.from(A.to);
    return ((a < 0 && b > 0) || (a > 0 && b < 0) || A.overlap(B.from) || A.
        overlap(B.to)) &&
        ((c < 0 && d > 0) || (c > 0 && d < 0) || B.overlap(A.from) || B.
            overlap(A.to));
}
```

## 平面几何通用

```
/// 计算几何专用. 按需选用.

db eps = 1e-12; // 线性误差范围; long double : 1e-16;
db eps2 = 1e-6; // 平方级误差范围; long double: 1e-8;
bool eq(db a, db b) { return abs(a-b) < eps; }

// ========================== 点和向量 ===========================
struct point;
struct point
{
    db x, y;
    point():x(0),y(0) { }
    point(db a,db b):x(a),y(b) { }
    point(point const& f):x(f.x),y(f.y) { }
    point operator=(point const& f) { x=f.x; y=f.y; return *this; }

    point operator+(point const& b) const { return point(x + b.x, y + b.y);
        }
    point operator-(point const& b) const { return point(x - b.x, y - b.y);
        }
    point operator()(point const& b) const { return b - *this; } // 从本顶点
        出发,指向另一个点的向量.

    db len2() const { return x*x+y*y; } // 模的平方.
    db len() const { return sqrt(len2()); } // 向量的模.
    point norm() const { db l = len(); return point(x/l, y/l); } // 标准化.

    // 把向量旋转f个弧度.
    point rot(double const& f) const
    { return point(x*cos(f) - y*sin(f), x*sin(f) + y*cos(f)); }

```

```
    // 极角，+x轴为0，弧度制，(- , ].
    db pangle() const { if(y >= 0) return acos(x / len()); else return -acos(x / len()); }

    void out() const { printf("(%.2f,␣%.2f)", x, y); } // 输出.
};

// 数乘.
point operator*(point const& a, db const& b) { return point(a.x * b, a.y * b); }
point operator*(db const& b, point const& a) { return point(a.x * b, a.y * b); }

// 叉积.
db operator*(point const& a, point const& b) { return a.x * b.y - a.y * b.x; }
// 点积.
db operator&(point const& a, point const& b) { return a.x * b.x + a.y * b.y; }

bool operator==(point const& a, point const& b) { return eq(a.x, b.x) && eq(a.y, b.y); }

// 判断本向量在另一个向量的顺时针方向. 注意选用eps或0.
bool operator>>(point const& a, point const& b) { return a*b > eps; }
// 判断本向量在另一个向量的顺时针方向或同向. 注意选用eps或0.
bool operator>>=(point const& a, point const& b) { return a*b > -eps; }

// ================================ 线段 ================================
struct segment
{
    point from,to;
    segment(point const& a = point(), point const& b = point()) : from(a), to(b) { }

    point dir() const { return to - from; } // 方向向量,未标准化.

    db len() const { return dir().len(); } // 长度.

    // 点在线段上.
    bool overlap(point const& v) const
    { return eq(from(to).len(), v(from).len() + v(to).len()); }

    point projection(point const& p) const // 点到直线上的投影.
    {
        db h = abs(dir() * from(p)) / len();
        db r = sqrt(from(p).len2() - h*h);
        if(eq(r, 0)) return from;
        if((from(to) & from(p)) < 0) return from + from(to).norm() * (-r);
        else return from + from(to).norm() * r;
    }
```

```
    point nearest(point const& p) const // 点到线段的最近点.
    {
        point g = projection(p);
        if(overlap(g)) return g;
        if(g(from).len() < g(to).len()) return from;
        return to;
    }
};

bool operator/(segment const& a, segment const& b) // 平行 (零向量平行于任意向量).
{
    return eq(a.dir() * b.dir(), 0);
}

// 相交. 不计线段端点则删掉 eq(..., 0) 的所有判断.
bool operator*(segment const& A, segment const& B)
{
    point dia = A.from(A.to);
    point dib = B.from(B.to);
    db a = dia * A.from(B.from);
    db b = dia * A.from(B.to);
    db c = dib * B.from(A.from);
    db d = dib * B.from(A.to);
    return ((a < 0 && b > 0) || (a > 0 && b < 0) || A.overlap(B.from) || A.overlap(B.to)) &&
        ((c < 0 && d > 0) || (c > 0 && d < 0) || B.overlap(A.from) || B.overlap(A.to));
}
```

## 立体几何通用

```
db eps = 1e-12; // 线性误差范围; long double : 1e-16;
db eps2 = 1e-6; // 平方级误差范围; long double: 1e-8;
bool eq(db a, db b) { return abs(a-b) < eps; }

// ======================== 点和向量 ========================
struct point;
struct point
{
    db x, y, z;
    point():x(0),y(0),z(0) { }
    point(db a,db b,db c):x(a),y(b),z(c) { }
    point(point const& f):x(f.x),y(f.y),z(f.z) { }
    point operator=(point const& f) { x=f.x; y=f.y; z=f.z; return *this; }

    point operator+(point const& b) const { return point(x + b.x, y + b.y, z + b.z); }
```

```cpp
    point operator-(point const& b) const { return point(x - b.x, y - b.y, z - b.z); }
    point operator()(point const& b) const { return b - *this; } // 从本顶点出发,指向另一个点的向量.

    db len2() const { return x*x+y*y+z*z; } // 模的平方.
    db len() const { return sqrt(len2()); } // 向量的模.
    point norm() const { db l = len(); return point(x/l, y/l, z/l); } // 标准化.

    void out(const char* c) const { printf("(%.2f, %.2f, %.2f)%s", x, y, z, c); } // 输出.
};

// 数乘.
point operator*(point const& a, db const& b) { return point(a.x * b, a.y * b, a.z * b); }
point operator*(db const& b, point const& a) { return point(a.x * b, a.y * b, a.z * b); }

// 叉积.
point operator*(point const& a, point const& b)
{ return point(a.y*b.z - a.z*b.y, a.z*b.x - a.x*b.z, a.x*b.y - a.y*b.x); }

// 点积.
db operator&(point const& a, point const& b)
{ return a.x * b.x + a.y * b.y + a.z * b.z; }

bool operator==(point const& a, point const& b)
{ return eq(a.x, b.x) && eq(a.y, b.y) && eq(a.z, b.z); }


// ============================== 线段 ==============================
struct segment
{
    point from,to;
    segment() : from(), to() { }
    segment(point const& a, point const& b) : from(a), to(b) { }

    point dir() const { return to - from; } // 方向向量,未标准化.
    db len() const { return dir().len(); } // 长度.
    db len2() const { return dir().len2(); }

    // 点在线段上.
    bool overlap(point const& v) const
    { return eq(from(to).len(), v(from).len() + v(to).len()); }

    point projection(point const& p) const // 点到直线上的投影.
    {
        db h2 = abs((dir() * from(p)).len2()) / len2();
        db r = sqrt(from(p).len2() - h2);
```

```
        if(eq(r, 0)) return from;
        if((from(to) & from(p)) < 0) return from + from(to).norm() * (-r);
        else return from + from(to).norm() * r;
    }

    point nearest(point const& p) const // 点到线段的最近点.
    {
        point g = projection(p);
        if(overlap(g)) return g;
        if(g(from).len() < g(to).len()) return from;
        return to;
    }

    point nearest(segment const& x) const // 线段x上的离本线段最近的点.
    {
        db l = 0.0, r = 1.0;
        while(r - l > eps)
        {
            db delta = r - l;
            db lmid = l + 0.4 * delta;
            db rmid = l + 0.6 * delta;
            point lp = x.interpolate(lmid);
            point rp = x.interpolate(rmid);
            point lnear = nearest(lp);
            point rnear = nearest(rp);
            if(lp(lnear).len2() > rp(rnear).len2()) l = lmid;
            else r = rmid;
        }
        return x.interpolate(l);
    }

    point interpolate(db const& p) const { return from + p * dir(); }
};

bool operator/(segment const& a, segment const& b) // 平行 (零向量平行于任意向量).
{
    return eq((a.dir() * b.dir()).len(), 0);
}
```

## 判断点在凸多边形内

```
/// 在线, 单次询问O(logn), st为凸包点数, 包括多边形上顶点和边界.
/// 要求凸包上没有相同点, 仅包含顶点.

bool agcmp(point const& a,point const& b) { return sp(a) * sp(b) < 0; }
bool PointInside(point target)
{
    sp = stk[0];
```

```
    point vt = sp(stk[1]);
    point vb = sp(stk[st-2]);
    db mt = vt * sp(target);
    db mb = vb * sp(target);
    bool able = (eq(mt, 0) && eq(mb, 0)) ||
        (eq(mt, 0) && mb > 0) || (eq(mb, 0) && mt < 0) ||
        (mt < 0 && mb > 0);
    if(able)
    {
        int xp = (int)(lower_bound(stk+1, stk+st-2, target, agcmp) - stk);
        able &= !(segment(sp, target) * segment(stk[xp], stk[xp-1]));
        able |= segment(stk[xp], stk[xp-1]).overlap(target);
    }
    return able;
}

/// 在线，单次询问O(logn)，st为凸包点数，**不**包括多边形上顶点和边界.

bool agcmp(point const& a,point const& b) { return sp(a) * sp(b) < 0; }
bool PointInside(point target)
{
    sp = stk[0];
    point vt = sp(stk[1]);
    point vb = sp(stk[st-2]);
    db mt = vt * sp(target);
    db mb = vb * sp(target);
    bool able = mt < 0 && mb > 0;
    if(able)
    {
        int xp = (int)(lower_bound(stk+1, stk+st-2, target, agcmp) - stk);
        able &= !(segment(sp, target) * segment(stk[xp], stk[xp-1]));
    }
    return able;
}
```

## 凸包

```
/// 凸包
/// 去除输入中重复顶点，保留头尾重复，顺时针顺序.

/// a: 输入点.
/// stk: 用来存凸包上的点的栈.
/// st: 栈顶下标，指向最后一个元素的下一个位置.
/// stk[0]: 凸包上 y 值最小的点中，x值最小的点.

////////////////////////////////////////////////////////////////////////////

int n;
point a[105000];
```

```
point stk[105000]; int st;

bool operator<(point const& a, point const& b) { return eq(a.y, b.y) ? a.x <
      b.x : a.y < b.y; }
// 使用 >> 则取凸包上的点.
// 使用 >>= 不取凸包上的点.
void Graham()
{
    sort(a,a+n);
    int g = (int)(unique(a, a+n) - a);
    st=0;

    for(int i=0;i<g;i++)
    {
        while(st>1 && (stk[st-2](stk[st-1]) >> stk[st-1](a[i]))) st--;
        stk[st++]=a[i];
    }
    int p=st;
    for(int i=g-2;i>=0;i--)
    {
        while(st>p && (stk[st-2](stk[st-1]) >> stk[st-1](a[i]))) st--;
        stk[st++]=a[i];
    }
}

/// [.] AC HDU 1392
```

## 旋转卡壳

```

/// 旋转卡壳求最远点对距离.
/// stk[]: 按顺序存储的凸壳上的点的数组.
////////////////////////////////////////////////////////////////////////////
int GetmaxDistance()
{
    int res=0;
    int p=2;
    for(int i=1;i<st;i++)
    {
        while( p!=st && area(stk[i-1], stk[i], stk[p+1]) > area(stk[i-1],
             stk[i], stk[p]))
            p++;
        // 此时stk[i]的对踵点是stk[p].
        if(p==st) break;
        // 修改至想要的部分.
        res=max(res,stk[i-1](stk[p]).dist2());
        res=max(res,stk[i](stk[p]).dist2());
    }
    return res;
```

```
}
```

## 最小覆盖圆

```
/// 最小覆盖圆.

/// n: 点数.
/// a: 输入点的数组.

///////////////////////////////////////////////////////////////////////////

const db eps = 1e-12;
const db eps2 = 1e-8;

/// 过三点的圆的圆心.
point CC(point const& a,point const& b,point const& c)
{
    point ret;
    db a1 = b.x-a.x, b1 = b.y-a.y, c1 = (a1*a1+b1*b1)*0.5;
    db a2 = c.x-a.x, b2 = c.y-a.y, c2 = (a2*a2+b2*b2)*0.5;
    db d = a1*b2 - a2*b1;
    if(abs(d)<eps) return (b+c)*0.5;
    ret.x=a.x+(c1*b2-c2*b1)/d;
    ret.y=a.y+(a1*c2-a2*c1)/d;
    return ret;
}

int n;
point a[1005000];

struct Resault{ db x,y,r; };
Resault MCC()
{
    if(n==0) return {0, 0, 0};
    if(n==1) return {a[0].x, a[0].y, 0};
    if(n==2) return {(a[0]+a[1]).x*0.5, (a[0]+a[1]).y*0.5, dist(a[0],a[1])
        *0.5};

    for(int i=0;i<n;i++) swap(a[i], a[rand()%n]); // 随机交换.

    point O; db R = 0.0;
    for(int i=2; i<n; i++) if(O(a[i]).len() >= R+eps2)
    {
        O=a[i];
        R=0.0;

        for(int j=0; j<i; j++) if(O(a[j]).len() >= R+eps2)
        {
            O=(a[i] + a[j]) * 0.5;
```

```
            R=a[i](a[j]).len() * 0.5;

            for(int k=0; k<j; k++) if(O(a[k]).len() >= R+eps2)
            {
                O = CC(a[i], a[j], a[k]);
                R = O(a[i]).len();
            }
        }
    }

    return {O.x, O.y, R};
}
```

# 数据结构

## KD 树

```

/// KD 树.
/// 最近邻点查询.
/// 维度越少剪枝优化效率越高. 4 维时是1/10倍运行时间, 8维时是1/3倍运行时间.
/// 板子使用欧几里得距离.
/// 可以把距离修改成曼哈顿距离之类的, **剪枝一般不会出错**.

////////////////////////////////////////////////////////////////

const int mxnc = 105000; // 最大的所有树节点数总量.
const int dem = 4; // 维度数量.

const db INF = 1e20;

/// 空间中的点.
struct point
{
    db v[dem]; // 维度坐标.
               // 注意你有可能用到每个维度坐标是不同的*类型*的点.
               // 此时需要写两个点对于第k个维度坐标的比较函数.
    point() { }
    point(db* coord) { memcpy(v, coord, sizeof(v)); }
    point(point const& x) { memcpy(v, x.v, sizeof(v)); }

    point& operator=(point const& x)
    { memcpy(v, x.v, sizeof(v)); return *this; }

    db& operator[](int const& k) { return v[k]; }
    db const& operator[](int const& k) const { return v[k]; }
};

```

```
db dist(point const& x, point const& y)
{
    db a = 0.0;
    for(int i=0; i<dem; i++) a += (x[i] - y[i]) * (x[i] - y[i]);
    return sqrt(a);
}

/// 树中的节点.
struct node
{
    point loc;  // 节点坐标点.
    int d;      // 该节点的下层节点从哪个维度切割. 切割坐标值由该节点坐标值
        给出.
    node* s[2]; // 左右子节点.

    int sep(point const& x) const { return x[d] >= loc[d]; }
};
node pool[mxnc]; node* curn = pool;

// 这个数组用来分配唯独切割顺序. 可以改用别的维度选择方式.
int flc[] = {3, 0, 2, 1};
node* newnode(point const& p, int dep)
{
    curn->loc = p;
    curn->d = flc[dep % dem];
    curn->s[0] = curn->s[1] = NULL;
    return curn++;
}

/// KD树.
struct KDTree
{
    node* root;

    KDTree() { root = NULL; }

    node* insert(point const& x)
    {
        node* cf = NULL;
        node* cur = root;
        int dep = 0;
        while(cur != NULL)
        {
            dep++;
            cf = cur;
            cur = cur->s[cur->sep(x)];
        }
        if(cf == NULL) return root = newnode(x, dep);
        return cf->s[cf->sep(x)] = newnode(x, dep);
    }

```

```
    // 求最近点的距离，以及最近点.
    pair<db, point*> nearest(point const& p, node* x)
    {
        if(x == NULL) return make_pair(INF, (point*)NULL);

        int k = x->sep(p);

        // 拿到点 p 从属子区域的结果.
        pair<db, point*> sol = nearest(p, x->s[k]);

        // 用当前区域存储的点更新答案.
        db cd = dist(x->loc, p);
        if(sol.first > cd)
        {
            sol.first = cd;
            sol.second = &(x->loc);
        }

        //  如果当前结果半径和另一个子区域相交，询问子区域并更新答案.
        db divDist = abs(p[x->d] - x->loc[x->d]);
        if(sol.first >= divDist)
        {
            pair<db, point*> solx = nearest(p, x->s[!k]);
            if(sol.first > solx.first) sol = solx;
        }

        return sol;
    }

    db nearestDist(point const& p) { return nearest(p, root).first; }
};

/// 初始化节点列表，会清除**所有树**的信息.
void Init()
{
    curn = pool;
}
```

## Splay

```
/// Splay.
///  没有特殊功能的平衡树. 预留了一个自底向上更新的update函数.
/// pool: 点的池子. Splay数据结构本身只保存根节点指针.
/// 重新初始化: nt = pool + 1; 不要更改nil.

/// mxn: 节点池子大小.
const int mxn = 205000;

////////////////////////////////////////////////////////////////////////////
```

```
struct node* nil;
struct node
{
    int v;
    int cnt;
    node*s[2];
    node*f;
    void update()
    {
        cnt=1;
        if(s[0]!=nil) cnt+=s[0]->cnt;
        if(s[1]!=nil) cnt+=s[1]->cnt;
    }
}
pool[mxn]; node* nt=pool;

node*newnode(int v, node*f)
{
    nt->v=v;
    nt->cnt=1;
    nt->s[0]=nt->s[1]=nil;
    nt->f=f;
    return nt++;
}


struct SplayTree
{
     node*root;
     SplayTree():root(nil){}

     void rot(node*x)
     {
          node*y=x->f;
          int k=(x==y->s[0]);

          y->s[k^1]=x->s[k];
          if(x->s[k]!=nil) x->s[k]->f=y;

          x->f=y->f;
          if(y->f!=nil) y->f->s[y==y->f->s[1]]=x;

          y->f=x; x->s[k]=y;

          y->update();
     }

     node* splay(node*x,node*t=nil)
     {
          while(x->f!=t)
```

```
        {
            node*y=x->f;
            if(y->f!=t)
            if((x==y->s[0])^(y==y->f->s[0]))
                rot(x); else rot(y);
            rot(x);
        }
        x->update();
        if(t==nil) root=x;
        return x;
    }

    //==============================================

    void Insert(int v)
    {
        if(root==nil) { root=newnode(v, nil); return; }
        node *x=root, *y=root;
        while(x!=nil) { y=x; x=x->s[x->v <= v]; }
        splay(y->s[y->v<=v] = newnode(v, y));
    }


    node*Find(int v) // 查找值相等的节点. 找不到会返回nil.
    {
        node *x=root, *y=root;
        node *r=nil;
        while(x!=nil)
        {
            y=x;
            if(x->v==v) r=x;
            x=x->s[x->v < v];
        }
        splay(y);
        return r;
    }

    node* FindRank(int k) // 查找排名为 k 的节点.
    {
        node *x=root, *y=root;
        while(x!=nil)
        {
            y=x;
            if(k==x->s[0]->cnt+1) break;
            if(k<x->s[0]->cnt+1) x=x->s[0];
            else { k-=x->s[0]->cnt+1; x=x->s[1]; }
        }
        splay(y);
        return x;
    }

```

```
// 排名从 1 开始.
int GetRank(node*x) { return splay(x)->s[0]->cnt+1; }

node*Delete(node*x)
{
    int k=GetRank(x);
    node*L=FindRank(k-1);
    node*R=FindRank(k+1);

    if(L!=nil) splay(L);
    if(R!=nil) splay(R,L);

    if(L==nil && R==nil) root=nil;
    else if(R==nil) L->s[1]=nil;
    else R->s[0]=nil;

    if(R!=nil) R->update();
    if(L!=nil) L->update();

    return x;
}

node*Prefix(int v) // 前驱.
{
    node *x=root, *y=root;
    node*r=nil;
    while(x!=nil)
    {
        y=x;
        if(x->v<v) r=x;
        x=x->s[x->v<v];
    }
    splay(y);
    return r;
}

node*Suffix(int v) // 后继.
{
    node *x=root, *y=root;
    node*r=nil;
    while(x!=nil)
    {
        y=x;
        if(x->v>v) r=x;
        x=x->s[x->v<=v];
    }
    splay(y);
    return r;
}

//==========================================
```

```
    void output() { output(root); printf("%s\n",root==nil ? "empty tree!" : ""); }
    void output(node*x)
    {
        if(x==nil)return ;
        output(x->s[0]);
        printf("%d ",x->v);
        output(x->s[1]);
    }

    void test() { test(root); printf("%s\n",root==nil ? "empty tree!" : ""); }
    void test(node*x)
    {
        if(x==nil)return ;
        test(x->s[0]);
        printf("%p [ v:%d f:%p L:%p R:%p cnt:%d ] \n",x,x->v,x->f,x->s[0],x->s[1],x->cnt);
        test(x->s[1]);
    }

};


int n;

int main()
{
    nil=newnode(-1, nullptr);
    nil->cnt=0;
    nil->f=nil->s[0]=nil->s[1]=nil;

    n=getint();
    SplayTree st;

    for(int i=0;i<n;i++)
    {
        int c;
        c=getint();
        switch(c)
        {
            case 1: //Insert
                c=getint();
                st.Insert(c);
            break;
            case 2: //Delete
                c=getint();
                st.Delete(st.Find(c));
            break;
            case 3: //Rank
                c=getint();
```

```
                printf("%d\n",st.GetRank(st.Find(c)));
            break;
            case 4: //FindRank
                c=getint();
                printf("%d\n",st.FindRank(c)->v);
            break;
            case 5: //prefix
                c=getint();
                printf("%d\n",st.Prefix(c)->v);
            break;
            case 6: //suffix
                 c=getint();
                 printf("%d\n",st.Suffix(c)->v);
            break;
            case 7: //test
                st.test();
            break;
            default: break;
        }
    }

    return 0;
}
```

## 表达式解析

```
/// 表达式解析
/// 线性扫描，直接计算.
/// 不支持三元运算符.
/// 一元运算符经过特殊处理. 它们不会(也不应)与二元运算符共用一种符号.

/// prio: 字符优先级. 在没有括号的约束下，优先级高的优先计算.
/// pref: 结合顺序. pref[i] == true 表示从左到右结合，false 则为从右到左结合
    .
/// 圆括号运算符会特别对待.

/// 如果需要建树，直接改Calc和Push函数.

/// ctt: 字符集编号下界.
/// ctf: 字符集编号上界.
/// ctx: 字符集大小.
const int ctf = -128;
const int ctt = 127;
const int ctx = ctt - ctf;

/// 表达式字符总数.
const int mxn = 1005000;

/// inp: 输入的表达式; 已经去掉了空格.
```

```
/// inpt: 输入的表达式的长度.
/// sx, aval: 由Destruct设定的外部变量数组. 无需改动.
/// 用法:
int len = Destruct(inp, inpt);
Evaluate(sx, len, aval);


/// 重新初始化: 调用Destruct即可.

////////////////////////////////////////////////////////////////////////

int _prio[ctx]; int* prio = _prio - ctf;
bool _pref[ctx]; bool* pref = _pref - ctf;

// 设置一个运算符的优先级和结合顺序.
void SetProp(char x, int a, int b) { prio[x] = a; pref[x] = b; }

stack<int> ap; // 变量栈.
stack<char> op; // 符号栈.

int Fetch() { int x = ap.top(); ap.pop(); return x; }
void Push(int x) { ap.push(x); }

/// 这个函数定义了如何处理栈内的实际元素.
void Calc()
{
    char cop = op.top(); op.pop();
    switch(cop)
    {
        case '+': { int b = Fetch(); int a = Fetch(); Push(a + b); } return;
        case '-': { int b = Fetch(); int a = Fetch(); Push(a - b); } return;
        case '*': { int b = Fetch(); int a = Fetch(); Push(a * b); } return;
        case '/': { int b = Fetch(); int a = Fetch(); Push(a / b); } return;
        case '|': { int b = Fetch(); int a = Fetch(); Push(a | b); } return;
        case '&': { int b = Fetch(); int a = Fetch(); Push(a & b); } return;
        case '^': { int b = Fetch(); int a = Fetch(); Push(a ^ b); } return;
        case '!': { int a = Fetch(); Push(a); } return;      // '+'的一元算符.
        case '~': { int a = Fetch(); Push(-a); } return;     // '-'的一元算符.
        default: return;
    }
}

/// s: 转化后的表达式, 其中0表示变量, 其它表示相应运算符. len: 表达式长度.
/// g: 变量索引序列, 表示表达式从左到右的变量分别是哪个.
void Evaluate(char* s, int len, int* g)
{
    int gc = 0;
    for(int i=0; i<len; i++)
    {
```

```
        if(s[i] == 0) // 输入是一个变量. 一般可以直接按需求改掉, 例如 if(IsVar(s[i])).
        {
            Push(g[gc++]); // 第gc个变量的**值**入栈.
        }
        else // 输入是一个运算符s[i].
        {
            if(s[i] == '(') op.push(s[i]);
            else if(s[i] == ')')
            {
                while(op.top() != '(') Calc();
                op.pop();
            }
            else
            {
                while( prio[s[i]] < prio[op.top()] ||
                    (prio[s[i]] == prio[op.top()] && pref[s[i]] == true))
                    Calc();
                op.push(s[i]);
            }
        }
    }
}

/// 解析一个字符串, 得到能够被上面的函数处理的格式.
/// 对于这个函数而言, "变量"是某个十进制整数.
/// 有些时候输入本身就是这样的格式, 就不需要过多处理.
/// 支持的二元运算符: +, -, *, /, |, &, ^. 支持的一元运算符: +, -.
char sx[mxn]; // 表达式序列.
int aval[mxn]; // 数字. 这些是扔到变量栈里面的东西.
               // 可以直接写成某种place holder, 如果不关心这些变量之间的区别的话.
/// 返回: 表达式序列长度.
int Destruct(char* s, int len)
{
    int xlen = 0;
    sx[xlen++] = '(';
    bool cvr = false;
    int x = 0;
    int vt = 0;
    for(int i=0; i<len; i++)
    {
        if('0' <= s[i] && s[i] <= '9')
        {
            if(!cvr) sx[xlen++] = 0;
            cvr = true;
            if(cvr) x = x * 10 + s[i] - '0';
        }
        else
        {
            if(cvr) { aval[vt++] = x; x = 0; }
```

```
            cvr = false;
            sx[xlen++] = s[i];
        }
    }
    if(cvr) { aval[vt++] = x; x = 0; }

    for(int i=xlen; i>=1; i--) // 一元运算符特判，修改成不同于二元运算符的符号.
        if((sx[i]=='+' || sx[i]=='-') && sx[i-1] != ')' && sx[i-1])
            sx[i] = sx[i] == '+' ? '!' : '~';

    sx[xlen++] = ')';
    return xlen;
}

char c[mxn];

char inp[mxn]; int inpt;
int main()
{
    SetProp('(', 0, true);
    SetProp(')', 0, true);

    SetProp('+', 10, true);
    SetProp('-', 10, true);

    SetProp('*', 100, true);
    SetProp('/', 100, true);

    SetProp('|', 1000, true);
    SetProp('&', 1001, true);
    SetProp('^', 1002, true);

    SetProp('!', 10000, false);
    SetProp('~', 10000, false);

    inpt = 0;
    char c;
    while((c = getchar()) != EOF && c != '\n' && c!='\r') if(c != ' ') inp[inpt++] = c;
    // 输入.
    printf("%s\n", inp);
    // 表达式符号.
    int len = Destruct(inp, inpt);
    for(int i=0; i<len; i++) if(sx[i] == 0) printf("."); else printf("%c", sx[i]); printf("\n");
    // 运算数.
    int t = 0; for(int i=0; i<len; i++) if(sx[i] == 0) printf("%d ", aval[t++]); printf("\n");
    Evaluate(sx, len, aval);
    // 结果.
```

```
    printf("%d\n", ap.top());

    return 0;
}

// (123+---213-+--321)+4*--57^6 = -159 correct!
```

## 并查

```
/// 并查集


/// 简易的集合合并并查集,带路径压缩.
/// 重新初始化:
memset(f, 0, sizeof(int) * (n+1));
//////////////////////////////////////////////////////////////////////////
int f[mxn];
int fidnf(int x){ return f[x]==x ? x : f[x]=findf(f[x]); }
int connect(int a,int b){ f[findf(a)]=findf(b); }


/// 集合并查集,带路径压缩和按秩合并.
/// c[i]: 点i作为集合表头时, 该集合大小.
/// 重新初始化:
memset(f, 0, sizeof(int) * (n+1));
memset(c, 0, sizeof(int) * (n+1));
//////////////////////////////////////////////////////////////////////////
int f[mxn];
int c[mxn];
int connect(int a,int b)
{
    if(c[findf(a)]>c[findf(b)]) // 把b接到a中.
    { c[findf(a)]+=c[findf(b)]; f[findf(b)] = findf(a); } // 执行顺序不可对调.
    else // 把a接到b中.
    { c[findf(b)]+=c[findf(a)]; f[findf(a)] = findf(b); }
}


/// 集合并查集,带路径压缩,非递归.
/// 重新初始化:
memset(f, 0, sizeof(int) * (n+1));
//////////////////////////////////////////////////////////////////////////
int f[mxn];
int findf(int x) // 传入参数x不可为引用.
{
    stack<int> q;
    while(f[x]!=x) q.push(x), x=f[x];
    while(!q.empty()) f[q.top()]=x, q.pop();
```

```
}
void connect(int a,int b){ f[findf(a)]=findf(b); } // *可以换成按秩合并版本*.
```

## 可持久化线段树

```
/// 可持久化线段树.

/// 动态开点的权值线段树; 查询区间k大;
/// 线段树节点记录区间内打上了标记的节点有多少个; 只支持插入; 不带懒标记.
/// 如果要打tag和推tag, 参考普通线段树. 注意这样做以后基本就不能支持两棵树相减.

/// 池子大小.
const int pg = 4000000;

/// 树根数量.
const int mxn = 105000;

/// 权值的最大值. 默认线段树的插入范围是 [0, INF].
const int INF=(1<<30)-1;

/// 重新初始化:
nt = 0;
...
SegmentTreeInit(n);

////////////////////////////////////////////////////////////////////////////

struct node
{
    int t;
    node*l,*r;
    node(){ t=0; l=r=NULL; }
    void update() { t=l->t+r->t; }
}pool[pg];

int nt;

node* newnode() { return &pool[nt++]; }

node* nil;
node* root[mxn];

void SegmentTreeInit(int size = 0)
{
    nil = newnode();
    nil->l = nil->r = nil;
    nil->t = 0;
```

```
    for(int i=0; i<=size; i++) root[i] = nil;
}

/// 在(子)树y的基础上新建(子)树x, 修改树中位置为cp的值.
int cp;
node*Change(node*x, node*y, int l = 0, int r = INF)
{
    if(cp<l || r<cp) return y;
    x=newnode();
    if(l==r) { x->t = 1 + y->t; return x; }
    int mid = (l+r)>>1;
    x->l = Change(x->l, y->l, l, mid);
    x->r = Change(x->r, y->r, mid+1, r);
    x->update();
    return x;
}

/// 查询树r减去树l的线段树中的第k大.
int Query(int ql,int qr,int k)
{
    node*x=root[ql],*y=root[qr];
    int l=0, r=INF;
    while(l != r)
    {
        int mid = (l+r)>>1;
        if(k <= x->l->t - y->l->t)
             r = mid, x = x->l,y = y->l;
        else
        {
            k -= x->l->t-y->l->t;
            l = mid+1, x = x->r, y = y->r;
        }
    }
    return l;
}

int n;

int main()
{

    int q;
    scanf("%d",&n);
    scanf("%d",&q);

    SegmentTreeInit(n);


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

```
        scanf("%d",&c);
        cp=c;
        root[i+1]=Change(root[i+1],root[i],0,INF);
    }


    for(int i=0;i<q;i++)
    {
        int a,b,k;
        scanf("%d%d%d",&a,&b,&k);
        printf("%d\n",Query(b,a-1,k));
    }

    return 0;
}
```

## 轻重边剖分

```
/// 轻重边剖分+dfs序.
const int mxn = 105000; // 最大节点数.

/// n: 实际点数.
/// c[i]: 顶点i属于的链的编号.
/// f[i]: 顶点i的父节点.
/// mxi[i]: 记录点i的重边应该连向哪个子节点.  用于dfs序构建.
/// sz[i]: 子树i的节点个数.
int n;
int c[mxn];
int f[mxn];
int mxi[mxn];
int sz[mxn];
/// ct: 链数.
/// ch[i]: 链头节点编号.
int ct;
int ch[mxn];
/// loc[i]: 节点i在dfs序中的位置.
/// til[i]: 子树i在dfs序中的末尾位置.
int loc[mxn];
int til[mxn];

/// 操作子树i的信息  <=> 操作线段树上闭区间 loc[i], til[i].
/// 操作路径信息 <=> 按照LCA访问方式访问线段树上的点.

/// 重新初始化:
et = pool;
for(int i=0; i<n; i++) eds[i] = NULL;

////////////////////////////////////////////////////////////////////////////

```

```cpp

struct edge{ int in; edge*nxt; } pool[mxn<<1];
edge*eds[mxn]; edge*et=pool;
void addedge(int a,int b){ et->in=b; et->nxt=eds[a]; eds[a]=et++; }
#define FOREACH_EDGE(e,x) for(edge*e=eds[x];e;e=e->nxt)
#define FOREACH_SON(e,x) for(edge*e=eds[x];e;e=e->nxt) if(f[x]!=e->in)

int q[mxn]; int qh,qt;
void BuildChain(int root) /// 拓扑序搜索(逆向广搜). 防爆栈.
{
    f[root]=-1; // 不要修改! 用于在走链时判断是否走到头了.
    q[qt++]=root;
    while(qh!=qt) { int x = q[qh++]; FOREACH_SON(e,x) { f[e->in] = x; q[qt++] = e->in;} }
    for(int i=n-1; i>=0; i--)
    {
        int x = q[i];
        sz[x] = 0;
        if(!eds[x]) { sz[x] = 1; ch[ct] = x; c[x] = ct++; continue; }
        int mxp = eds[x]->in;
        FOREACH_SON(e,x)
        {
            sz[x] += sz[e->in];
            if(sz[e->in] > sz[mxp]) mxp = e->in;
        }
        c[x] = c[mxi[x] = mxp]; ch[c[x]] = x;
    }
}

// 如果不需要dfs序, 只需要节点所在链的信息, 该函数可以放空.
int curl;
void BuildDFSOrder(int x)
{
    loc[x] = curl++;
    if(eds[x]) BuildDFSOrder(mxi[x]); // dfs序按照重边优先顺序构造, 可以保证所有重边在dfs序上连续.
    FOREACH_SON(e,x) if(e->in != mxi[x]) BuildDFSOrder(e->in);
    til[x] = curl-1;
}

void HLD(int root)
{
    ct = 0;
    BuildChain(root);
    curl = 0;
    BuildDFSOrder(root);
}

/// 线段树.
#define L (x<<1)
#define R (x<<1|1)
```

```
int t[mxn<<3];
int tag[mxn<<3];

inline void pushtag(int x,int l,int r)
{
    if(tag[x]==0) return;
    tag[L] = tag[R] = tag[x];
    int mid = (l+r)>>1;
    if(tag[x]==-1) { t[L]=t[R]=0; }
    else if(tag[x]==1) { t[L]=mid-l+1; t[R]=r-mid; }
    tag[x]=0;
}
inline void Update(int x,int l,int r)
{ t[x] = t[L] + t[R]; }

int cl, cr, cv;
void Change(int x=1, int l=0, int r=n-1)
{
    if(cr<l || r<cl) return;
    if(cl<=l && r<=cr)
        { tag[x] = cv; t[x] = ( tag[x]==-1 ? 0 : r-l+1 ); return; }
    pushtag(x,l,r);
    int mid = (l+r)>>1;
    Change(L,l,mid); Change(R,mid+1,r); Update(x,l,r);
}
void Modify(int l,int r,int v) { cl=l; cr=r; cv=v; Change(); }

int ql,qr;
int Query(int x=1, int l=0, int r=n-1)
{
    pushtag(x,l,r);
    if(qr<l || r<ql) return 0;
    if(cl<=l && r<=cr) return t[x];
    int mid = (l+r)>>1;
    return Query(L,l,mid) + Query(R,mid+1,r);
}
int GetTotalSum() { return t[1]; }

/// 修改到根的路径上的信息. 按需更改.
void Install(int p)
{
    do{
        Modify(loc[ch[c[p]]], loc[p], 1);
        p=f[ch[c[p]]];
    }
    while(p!=-1);
}

/// 修改子树信息. 按需更改.
void Remove(int p)
{
```

```
    Modify(loc[p], til[p], -1);
}
```

## 手写 bitset

```
/*
    预处理p[i] = 2^i
    保留N位
    get(d)获取d位
    set(d,x)将d位设为x
    count()返回1的个数
    zero()返回是不是0
    print()输出
*/
#define lsix(x) ((x)<<6)
#define rsix(x) ((x)>>6)
#define msix(x) ((x)-(((x)>>6)<<6))
ull p[64] = {1};
struct BitSet{
    ull s[rsix(N-1)+1];
    int cnt;
    void resize(int n){
        if(n>N)n=N;
        int t = rsix(n-1)+1;
        if(cnt<t)
            memset(s+cnt,0,sizeof(ull)*(t-cnt));
        cnt = t;
    }
    BitSet(int n){
        SET(s,0);
        cnt=1;
        resize(n);
    }
    BitSet(){cnt=1;SET(s,0);}
    BitSet operator & (BitSet &that){
        int len = min(that.cnt, this->cnt);
        BitSet ans(lsix(len));
        Repr(i,len)ans.s[i] = this->s[i] & that.s[i];
        ans.maintain();
        return ans;
    }
    BitSet operator | (BitSet &that){
        int len = max(that.cnt, this->cnt);
        BitSet ans(lsix(len));
        Repr(i,len)ans.s[i] = this->s[i] | that.s[i];
        ans.maintain();
        return ans;
    }
    BitSet operator ^ (BitSet &that){
```

```
        int len = max(that.cnt, this->cnt);
        BitSet ans(lsix(len));
        Repr(i,len)ans.s[i] = this->s[i] ^ that.s[i];
        ans.maintain();
        return ans;
    }
    BitSet operator << (int x){
        int c = rsix(x), r = msix(x);
        BitSet ans(lsix(cnt+c+(r!=0)));
        for (int i = min(ans.cnt-1, cnt+c);i-c >= 0; --i){
            if(i-c<cnt)
                ans.s[i] = s[i-c] << r;
            if (r && i-c-1 >= 0) ans.s[i] |= s[i-c-1] >> (64-r);
        }
        ans.maintain();
        return ans;
    }
    BitSet operator >> (int x){
        int c = rsix(x), r = msix(x);
        BitSet ans(lsix(cnt));
        if(c>=cnt)return ans;
        Rep(i,cnt-c){
            ans.s[i] = s[i+c] >> r;
            if (r && i+c+1 < cnt) ans.s[i] |= s[i+c+1] << (64-r);
        }
        ans.maintain();
        return ans;
    }
    int get(int d){
        int c = rsix(d), r = msix(d);
        if(c>=cnt)return 0;
        return (s[c] & p[r]);
    }
    void set(int d, int x){
        if(d>N)return;
        int c = rsix(d), r = msix(d);
        if(c>=cnt)
            resize(lsix(c+1));
        if(x&&(s[c] & p[r]))return;
        if(!x&&!(s[c] & p[r]))return;
        s[c] ^= p[r];
    }
    int count(){
        int res=0;
        Rep(i,cnt){
            ull x = s[i];
            while(x){
                res++;
                x&=x-1;
            }
        }
```

```
        return res;
    }
    void maintain(){
        while(s[cnt-1]==0&&cnt>1)
            cnt--;
        if(lsix(cnt)>N){
            while(lsix(cnt)>N)cnt--;
            if(lsix(cnt)<N){
                cnt++;
                for(int i = 63;i>N-lsix(cnt-1)-1;--i)
                    if(p[i]&s[cnt-1])s[cnt-1]-=p[i];
            }
        }
    }
    bool zero(){
        Rep(i,cnt)if(s[i])return 0;
        return 1;
    }
    void print(){
        if(lsix(cnt)<=N){
            rep(i,N-lsix(cnt))putchar('0');
            Repr(j,64)putchar(p[j] & s[cnt-1]?'1':'0');
        }else{
            Repr(i,N-lsix(cnt-1)-1)
                putchar(p[i] & s[cnt-1]?'1':'0');
        }
        Repr(i,cnt-2){
            ull x = s[i];
            Repr(j,64)putchar(p[j] & x?'1':'0');
        }
        putchar('\n');
    }
};
```

## 树状数组

```
inline int lowbit(int x){return x&-x;}
//前缀和,可改前缀最值
void update(int d, int x=1){
    if(!d)return;
    while(d<=n){
        T[d]+=x;
        d+=lowbit(d);
    }
}
int ask(int d){
    int res(0);
    while(d>0){
        res+=T[d];
```

```
        d-=lowbit(d);
    }
    return res;
}
```

## 线段树

```
/// 线段树.
/// 带乘法和加法标记.
/// 只作为样例解释.

/// mxn: 区间节点数. 线段树点数是它的四倍.
const int mxn = 105000;
/// n: 实际节点数.
/// a: 初始化列表.

/// 重新初始化:
build(); // 可以不使用初始化数组A.

//////////////////////////////////////////////////////////////////////////

ll a[mxn];
int n,m;
ll MOD;

#define L (x<<1)
#define R (x<<1|1)
ll t[mxn<<2]; // 当前真实值.
ll tagm[mxn<<2]; // 乘法标记.
ll taga[mxn<<2]; // 加法标记. 在乘法之后应用.
void pushtag(int x,int l,int r)
{
    if(tagm[x]==1 && taga[x]==0) return;
    ll &m = tagm[x]; ll &a = taga[x];
    // 向下合并标记.
    (tagm[L] *= m) %= MOD;
    (tagm[R] *= m) %= MOD;
    taga[L] = (taga[L] * m % MOD + a) % MOD;
    taga[R] = (taga[R] * m % MOD + a) % MOD;
    // 修改子节点真实值.
    int mid = (l+r)>>1;
    t[L] = (t[L] * m % MOD + (mid-l+1) * a) % MOD;
    t[R] = (t[R] * m % MOD + (r-mid) * a) % MOD;
    // 清理当前标记.
    tagm[x] = 1;
    taga[x] = 0;
}

/// 从子节点更新当前节点真实值.
```

```
/// 以下程序可以保证在Update之前该节点已经没有标记.
void update(int x) { t[x] = (t[L] + t[R]) % MOD; }

void build(int x=1,int l=1,int r=n) // 初始化.
{
    taga[x] = 0; tagm[x] = 1;
    if(l==r) { t[x] = a[l] % MOD; return; }
    int mid=(l+r)>>1;
    build(L,l,mid); build(R,mid+1,r);
    update(x);
}

int cl,cr; ll cv; int ct;
void Change(int x=1,int l=1,int r=n)
{
    if(cr<l || r<cl) return;
    if(cl<=l && r<=cr) // 是最终访问节点, 修改真实值并打上标记.
    {
        if(ct == 1)
        {
            (tagm[x] *= cv) %= MOD;
            (taga[x] *= cv) %= MOD;
            (t[x] *= cv) %= MOD;
        }
        else if(ct == 2)
        {
            (taga[x] += cv) %= MOD;
            (t[x] += (r-l+1) * cv) %= MOD;
        }
        return;
    }
    pushtag(x,l,r); // 注意不要更改推标记操作的位置.
    int mid = (l+r)>>1;
    Change(L,l,mid); Change(R,mid+1,r); update(x);
}

void Modify(int l,int r,ll v,int type)
{ cl=l; cr=r; cv=v; ct=type; Change(); }

int ql,qr;
ll Query(int x=1,int l=1,int r=n)
{
    if(qr<l || r<ql) return 0;
    if(ql<=l && r<=qr) return t[x];
    pushtag(x,l,r); // 注意不要更改推标记操作的位置.
    int mid=(l+r)>>1;
    return (Query(L,l,mid) + Query(R,mid+1,r)) % MOD;
}
ll Getsum(int l,int r)
{ ql=l; qr=r; return Query(); }
```

```
void Output(int x=1,int l=1,int r=n,int depth=0)
{
    printf("[%d] [%d,%d] t:%lld m:%lld a:%lld\n",x,l,r,t[x],taga[x],tagm[x])
        ;
    if(l==r) return;
    int mid=(l+r)>>1;
    Output(L,l,mid); Output(R,mid+1,r);
}

int main()
{
    n=getint(); MOD=getint();
    for(int i=1;i<=n;i++) a[i]=getint();
    build();
    m=getint();
    for(int i=0;i<m;i++)
    {
        int type = getint();
        if(type==3)
        {
            int l = getint();
            int r = getint();
            printf("%lld\n",Getsum(l,r));
        }
        else
        {
            int l = getint();
            int r = getint();
            int v = getint();
            Modify(l,r,v,type);
        }
    }
    return 0;
}
```

## 左偏树

```
int n,m,root,add;
struct node{
    int key,l,r,fa,add;
}heap1[maxn*2+1],heap2[maxn*2+1];
void down(int x){
    heap1[heap1[x].l].key+=heap1[x].add;
    heap1[heap1[x].l].add+=heap1[x].add;
    heap1[heap1[x].r].key+=heap1[x].add;
    heap1[heap1[x].r].add+=heap1[x].add;
    heap1[x].add=0;
}
int fa(int x){
```

```
    int tmp=x;
    while (heap1[tmp].fa) tmp=heap1[tmp].fa;
    return tmp;
}
int sum(int x){
    int tmp=x,sum=0;
    while (tmp=heap1[tmp].fa) sum+=heap1[tmp].add;
    return sum;
}
int merge1(int x,int y){
    if (!x || !y) return x?x:y;
    if (heap1[x].key<heap1[y].key) swap(x,y);
    down(x);
    heap1[x].r=merge1(heap1[x].r,y);
    heap1[heap1[x].r].fa=x;
    swap(heap1[x].l,heap1[x].r);
    return x;
}
int merge2(int x,int y){
    if (!x || !y) return x?x:y;
    if (heap2[x].key<heap2[y].key) swap(x,y);
    heap2[x].r=merge2(heap2[x].r,y);
    heap2[heap2[x].r].fa=x;
    swap(heap2[x].l,heap2[x].r);
    return x;
}
int del1(int x){
    down(x);
    int y=merge1(heap1[x].l,heap1[x].r);
    if (x==heap1[heap1[x].fa].l) heap1[heap1[x].fa].l=y;else heap1[heap1[x].fa].r=y;
    heap1[y].fa=heap1[x].fa;
    return fa(y);
}
void del2(int x){
    int y=merge2(heap2[x].l,heap2[x].r);
    if (root==x) root=y;
    if (x==heap2[heap2[x].fa].l) heap2[heap2[x].fa].l=y;else heap2[heap2[x].fa].r=y;
    heap2[y].fa=heap2[x].fa;
}
void renew1(int x,int v){
    heap1[x].key=v;
    heap1[x].fa=heap1[x].l=heap1[x].r=0;
}
void renew2(int x,int v){
    heap2[x].key=v;
    heap2[x].fa=heap2[x].l=heap2[x].r=0;
}
//建树
int heapify(){
```

```
    queue<int> Q;
    for (int i=1;i<=n;++i) Q.push(i);
    while (Q.size()>1){
        int x=Q.front();Q.pop();
        int y=Q.front();Q.pop();
        Q.push(merge2(x,y));
    }
    return Q.front();
}
//合并两棵树
void U(){
    int x,y;scanf("%d%d",&x,&y);
    int fx=fa(x),fy=fa(y);
    if (fx!=fy) if (merge1(fx,fy)==fx) del2(fy);else del2(fx);
}
//单点修改
void A1(){
    int x,v;scanf("%d%d",&x,&v);
    del2(fa(x));
    int y=del1(x);
    renew1(x,heap1[x].key+v+sum(x));
    int z=merge1(y,x);
    renew2(z,heap1[z].key);
    root=merge2(root,z);
}
//联通块修改
void A2(){
    int x,v,y;scanf("%d%d",&x,&v);
    del2(y=fa(x));
    heap1[y].key+=v;
    heap1[y].add+=v;
    renew2(y,heap1[y].key);
    root=merge2(root,y);
}
//全局修改
void A3(){
    int v;scanf("%d",&v);
    add+=v;
}
//单点查询
void F1(){
    int x;scanf("%d",&x);
    printf("%d\n",heap1[x].key+sum(x)+add);
}
//联通块最大值
void F2(){
    int x;scanf("%d",&x);
    printf("%d\n",heap1[fa(x)].key+add);
}
//全局最大值
void F3(){
```

```
    printf("%d\n",heap2[root].key+add);
}
int main(){
    scanf("%d",&n);
    for (int i=1;i<=n;++i)
        scanf("%d",&heap1[i].key),heap2[i].key=heap1[i].key;
    root=heapify();
    scanf("%d",&m);
    for (int i=1;i<=m;++i){
        scanf("%s",s);
        if (s[0]=='U') U();
        if (s[0]=='A'){
            if (s[1]=='1') A1();
            if (s[1]=='2') A2();
            if (s[1]=='3') A3();
        }
        if (s[0]=='F'){
            if (s[1]=='1') F1();
            if (s[1]=='2') F2();
            if (s[1]=='3') F3();
        }
    }
    return 0;
}
```

# 动态规划

## 插头 DP

```
//POJ 2411
//一个row*col的矩阵，希望用2*1或者1*2的矩形来填充满，求填充的总方案数
//输入为长和宽
#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;
#define LL long long

const int maxn=2053;
struct Node
{
    int H[maxn];
    int S[maxn];
    LL N[maxn];
    int size;
    void init()
    {
```

```
        size=0;
        memset(H,-1,sizeof(H));
    }
    void push(int SS,LL num)
    {
        int s=SS%maxn;
        while( ~H[s] && S[H[s]]!=SS )
            s=(s+1)%maxn;

        if(~H[s])
        {
            N[H[s]]+=num;
        }
        else
        {
            S[size]=SS;
            N[size]=num;
            H[s]=size++;
        }
    }
    LL get(int SS)
    {
        int s=SS%maxn;
        while( ~H[s] && S[H[s]]!=SS )
            s=(s+1)%maxn;

        if(~H[s])
        {
            return N[H[s]];
        }
        else
        {
            return 0;
        }
    }
} dp[2];
int now,pre;
int get(int S,int p,int l=1)
{
    if(p<0) return 0;
    return (S>>(p*l))&((1<<l)-1);
}
void set(int &S,int p,int v,int l=1)
{
    S^=get(S,p,l)<<(p*l);
    S^=(v&((1<<l)-1))<<(p*l);
}
int main()
{
    int n,m;
    while( scanf("%d%d",&n,&m),n||m )
```

```
    {
        if(n%2 && m%2) {puts("0");continue;}
        int now=1,pre=0;
        dp[now].init();
        dp[now].push(0,1);
        for(int i=0;i<n;i++) for(int j=0;j<m;j++)
        {
            swap(now,pre);
            dp[now].init();
            for(int s=0;s<dp[pre].size;s++)
            {
                int S=dp[pre].S[s];
                LL num=dp[pre].N[s];
                int p=get(S,j);
                int q=get(S,j-1);
                int nS=S;
                set(nS,j,1-p);
                dp[now].push(nS,num);
                if(p==0 && q==1)
                {
                    set(S,j-1,0);
                    dp[now].push(S,num);
                }
            }
        }
        printf("%lld\n",dp[now].get(0));
    }
}
```

## 概率 DP

```
/*
POJ 2096

一个软件有s个子系统，会产生n种bug
某人一天发现一个bug,这个bug属于一个子系统，属于一个分类
每个bug属于某个子系统的概率是1/s,属于某种分类的概率是1/n
问发现n种bug,每个子系统都发现bug的天数的期望。

dp[i][j]表示已经找到i种bug,j个系统的bug，达到目标状态的天数的期望
dp[n][s]=0;要求的答案是dp[0][0];
dp[i][j]可以转化成以下四种状态:
    dp[i][j],发现一个bug属于已经有的i个分类和j个系统。概率为(i/n)*(j/s);
    dp[i][j+1],发现一个bug属于已有的分类，不属于已有的系统.概率为 (i/n)*(1-j/s);
    dp[i+1][j],发现一个bug属于已有的系统，不属于已有的分类,概率为 (1-i/n)*(j/s);
    dp[i+1][j+1],发现一个bug不属于已有的系统，不属于已有的分类,概率为 (1-i/n)*(1-j/s);
```

```
整理便得到转移方程
*/

#include<stdio.h>
#include<iostream>
#include<algorithm>
#include<string.h>
using namespace std;
const int MAXN = 1010;
double dp[MAXN][MAXN];

int main()
{
    int n, s;
    while (scanf("%d%d", &n, &s) != EOF)
    {
        dp[n][s] = 0;
        for (int i = n; i >= 0; i--)
            for (int j = s; j >= 0; j--)
            {
                if (i == n && j == s)continue;
                dp[i][j] = (i * (s - j) * dp[i][j + 1] + (n - i) * j * dp[i + 1][j] + (n - i) * (s - j) * dp[i + 1][j + 1] + n * s) / (n * s - i * j);
            }
        printf("%.4lf\n", dp[0][0]);
    }
    return 0;
}
```

## 数位 DP

```
//HDU-2089 输出不包含4和62的数字的个数

int dp[10][10];
int k = 0;
int dig[100];

void init()
{
    dp[0][0] = 1;
    for (int i = 1; i <= 7; i++){
        for (int j = 0; j < 10; j++){
            for (int k = 0; k < 10; k++){
                if (j != 4 && !(j == 6 && k == 2)){
                    dp[i][j] += dp[i - 1][k];
                }
            }
        }
```

```
    }
}

int  solve (int num)
{
    int ret = num, ans = 0;
    memset(dig, 0, sizeof(dig));
    k = 1;
    while (ret > 0)
    {
        dig[k++] = ret % 10;
        ret /= 10;
    }
    for (int i = k; i > 0; i--)
    {
        for (int j = 0; j < dig[i]; j++)
        {
            if (!(j == 2 && dig[i + 1] == 6) && j != 4)
            {
                ans += dp[i][j];
            }
        }
        if (dig[i] == 4 || (dig[i] == 2 && dig[i + 1] == 6))
        {
            break;
        }
    }
    return ans;
}

int main() {
    int n, m;
    init();
    while (cin >> n >> m && (n + m))
    {
        int ans = solve(m + 1) - solve(n);
        cout << ans << endl;
    }
    return 0;
}
```

## 四边形 DP

```
/*HDOJ2829
题目大意：给定一个长度为n的序列，至多将序列分成m段，每段序列都有权值，权值为
    序列内两个数两两相乘之和。m<=n<=1000. 令权值最小。
状态转移方程：
dp[c][i]=min(dp[c][i],dp[c-1][j]+w[j+1][i])
url->:http://blog.csdn.net/bnmjmz/article/details/41308919
```

```
*/

#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
const int INF = 1 << 30;
const int MAXN = 1000 + 10;
typedef long long LL;
LL dp[MAXN][MAXN];//dp[c][j]表示前j个点切了c次后的最小权值
int val[MAXN];
int w[MAXN][MAXN];//w[i][j]表示i到j无切割的权值
int s[MAXN][MAXN];//s[c][j]表示前j个点切的第c次的位置
int sum[MAXN];
int main()
{
    int n, m;
    while (~scanf("%d%d", &n, &m))
    {
        if (n == 0 && m == 0)break;
        memset(s, 0, sizeof(s));
        memset(w, 0, sizeof(w));
        memset(dp, 0, sizeof(dp));
        memset(sum, 0, sizeof(sum));
        for (int i = 1; i <= n; ++i)
        {
            scanf("%d", &val[i]);
            sum[i] += sum[i - 1] + val[i];
        }
        for (int i = 1; i <= n; ++i)
        {
            w[i][i] = 0;
            for (int j = i + 1; j <= n; ++j)
            {
                w[i][j] = w[i][j - 1] + val[j] * (sum[j - 1] - sum[i - 1]);
            }
        }
        for (int i = 1; i <= n; ++i)
        {
            for (int j = 1; j <= m; ++j)
            {
                dp[j][i] = INF;
            }
        }
        for (int i = 1; i <= n; ++i)
        {
            dp[0][i] = w[1][i];
            s[0][i] = 0;
        }
        for (int c = 1; c <= m; ++c)
        {
```

```
            s[c][n + 1] = n; //设置边界
            for (int i = n; i > c; --i)
            {
                int tmp = INF, k;
                for (int j = s[c - 1][i]; j <= s[c][i + 1]; ++j)
                {
                    if (dp[c - 1][j] + w[j + 1][i] < tmp)
                    {
                        tmp = dp[c - 1][j] + w[j + 1][i]; //状态转移方程，j之前切了c-1次，第c次切j到j+1间的
                        k = j;
                    }
                }
                dp[c][i] = tmp;
                s[c][i] = k;
            }
        }
        printf("%d\n", dp[m][n]);
    }
    return 0;
}
```

## 完全背包

```
for (int i = 1;i <= N;i++){
    for (int v = weight[i];v <= V;v++){
        f[v] = max(f[v],f[v - weight[i]] + Value[i]);
    }
}
```

## 斜率 DP

```
//HDU 3507
//给出n,m，求在n个数中分成任意段，每段的花销是(sigma(a[l],a[r])+m)^2，求最小值
//http://acm.hdu.edu.cn/showproblem.php?pid=3507

#include <stdio.h>
#include <iostream>
#include <string.h>
#include <queue>
using namespace std;
const int MAXN = 500010;

int dp[MAXN];
int q[MAXN];
int sum[MAXN];
```

```
int head, tail, n, m;

int getDP(int i, int j)
{
    return dp[j] + m + (sum[i] - sum[j]) * (sum[i] - sum[j]);
}

int getUP(int j, int k)
{
    return dp[j] + sum[j] * sum[j] - (dp[k] + sum[k] * sum[k]);
}
int getDOWN(int j, int k)
{
    return 2 * (sum[j] - sum[k]);
}

int main()
{
    while (scanf("%d%d", &n, &m) == 2)
    {
        for (int i = 1; i <= n; i++)
            scanf("%d", &sum[i]);
        sum[0] = dp[0] = 0;
        for (int i = 1; i <= n; i++)
            sum[i] += sum[i - 1];
        head = tail = 0;
        q[tail++] = 0;
        for (int i = 1; i <= n; i++)
        {
            while (head + 1 < tail && getUP(q[head + 1], q[head]) <= sum[i]*getDOWN(q[head + 1], q[head]))
                head++;
            dp[i] = getDP(i, q[head]);
            while (head + 1 < tail && getUP(i, q[tail - 1])*getDOWN(q[tail - 1], q[tail - 2]) <= getUP(q[tail - 1], q[tail - 2])*getDOWN(i, q[tail - 1]))
                tail--;
            q[tail++] = i;
        }
        printf("%d\n", dp[n]);
    }
    return 0;
}
```

## 状压 DP

```
//CF 580D
```

```
//有n种菜，选m种。每道菜有一个权值，有些两个菜按顺序挨在一起会有combo的权值加成。求最大权值

#include <bits/stdc++.h>
using namespace std;
const int maxn = 20;
typedef long long LL;

int a[maxn];
int comb[maxn][maxn];
LL dp[(1 << 18) + 10][maxn];
LL ans = 0;
int n, m, k;

int Cnt(int st)
{
    int res = 0;
    for (int i = 0; i < n; i++)
    {
        if (st & (1 << i))
        {
            res++;
        }
    }
    return res;
}

int main()
{
    memset(comb, 0, sizeof comb);
    scanf("%d%d%d", &n, &m, &k);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }
    for (int i = 0; i < k; i++)
    {
        int x, y, c;
        scanf("%d%d%d", &x, &y, &c);
        x--;
        y--;
        comb[x][y] = c;
    }
    int end = (1 << n);
    memset(dp, 0, sizeof dp);
    for (int st = 0; st < end; st++)
    {
        for (int i = 0; i < n; i++)
        {
            if (st & (1 << i))
            {
```

```
                bool has = false;
                for (int j = 0; j < n; j++)
                {
                    if (j != i && (st & (1 << j)))
                    {
                        has = true;
                        dp[st][i] = max(dp[st][i], dp[st ^ (1 << i)][j] + a[
                            i] + comb[j][i]);
                    }
                }
                if (!has)
                {
                    dp[st][i] = a[i];
                }
            }
            if (Cnt(st) == m)
            {
                ans = max(ans, dp[st][i]);
            }
        }
    }

    cout << ans << endl;
    return 0;
}
```

## 最长上升子序列

```
//使用lisDP查找,a为待查找串,b用于返回结果串,n为a的长度
int dpSearch(int num, int low, int high)
{
    int mid;
    while (low <= high)
    {
        mid = (low + high) / 2;
        if (num >= b[mid]) low = mid + 1;
        else high = mid - 1;
    }
    return low;
}

int lisDP(int* a,int* b,int n)
{
    int i, len, pos;
    b[1] = a[1];
    len = 1;
    for (i = 2; i <= n; i++)
    {
        if (a[i] >= b[len])
```

```
        {
            len = len + 1;
            b[len] = a[i];
        }
        else
        {
            pos = dpSearch(a[i], 1, len);
            b[pos] = a[i];
        }
    }
    return len;
}
```

# 图论

## best's therom

```
/*
    以某个点为起点的欧拉回路数=该点为根的树形图数*(所有点出度-1)的乘积
    从1出发的欧拉回路的数量
    重边当作多种方案
*/
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <map>
#include <queue>
#include <set>
#include <stack>
#include <string>
#include <vector>

#define each(i, n) for (int(i) = 0; (i) < (n); (i)++)
#define reach(i, n) for (int(i) = n - 1; (i) >= 0; (i)--)
#define range(i, st, en) for (int(i) = (st); (i) <= (en); (i)++)
#define rrange(i, st, en) for (int(i) = (en); (i) >= (st); (i)--)
#define fill(ary, num) memset((ary), (num), sizeof(ary))

using namespace std;
typedef long long ll;

const int maxn = 410;
const int mod = 998244353;

int d[maxn][maxn], g[maxn][maxn];
ll c[maxn][maxn];
```

```
int in[maxn], mul[(int)2e5 + 10], out[maxn];

int n;

ll getDet(ll a[][maxn], int n)
{
    range(i, 1, n) range(j, 1, n) a[i][j] = (a[i][j] + mod) % mod;
    ll ret = 1;
    range(i, 2, n)
    {
        range(j, i + 1, n) while (a[j][i])
        {
            ll t = a[i][i] / a[j][i];
            range(k, i, n) a[i][k] = (a[i][k] - a[j][k] * t % mod + mod) %
                mod;
            range(k, i, n) swap(a[i][k], a[j][k]);
            ret = -ret;
        }
        if (a[i][i] == 0)
            return 0;
        ret = ret * a[i][i] % mod;
    }
    return (ret + mod) % mod;
}

ll fastPow(ll n, ll m)
{
    ll ans = 1;
    while (m) {
        if (m & 1)
            ans = ans * n % mod;
        n = n * n % mod;
        m >>= 1;
    }
    return ans;
}

bool judgeEuler()
{
    range(i, 1, n) if (in[i] != out[i]) return false;
    return true;
}

int main()
{
    int cas = 0;
    mul[0] = mul[1] = 1;
    range(i, 2, (int)(2e5 + 5)) mul[i] = (mul[i - 1] * 1LL * i) % mod;
    while (scanf("%d", &n) != EOF) {
        fill(in, 0), fill(d, 0), fill(out, 0);
        range(i, 1, n) range(j, 1, n)
```

```
            {
                scanf("%d", &g[i][j]);
                d[j][j] += g[i][j];
                in[j] += g[i][j];
                out[i] += g[i][j];
            }
            if (!judgeEuler()) {
                printf("Case #%d: 0\n", ++cas);
                continue;
            } else if (n == 1) {
                printf("Case #%d: %d\n", ++cas, mul[g[1][1]]);
                continue;
            }
            range(i, 1, n) range(j, 1, n) c[i][j] = d[i][j] - g[i][j];
            ll trees = getDet(c, n) % mod * mul[in[1]] % mod;
            range(i, 2, n) trees = trees * mul[in[i] - 1] % mod;
            range(i, 1, n) range(j, 1, n) trees = trees * fastPow(mul[g[i][j]],
                mod - 2) % mod;
            printf("Case #%d: %lld\n", ++cas, trees);
        }
        return 0;
}
/*
    欧拉回路：每条边恰走一次的回路
    欧拉通路：每条边恰走一次的路径
    欧拉图：存在欧拉回路的图
    半欧拉图：存在欧拉通路的图
    有向欧拉图：每个点入度=出度
    无向欧拉图：每个点度数为偶数
    有向半欧拉图：一个点入度=出度+1，一个点入度=出度-1，其他点入度=出度
    无向半欧拉图：两个点度数为奇数，其他点度数为偶数
*/
```

## k 短路可持久化堆

```
/*
    s到t的k短路
*/
typedef long long LL ;
typedef pair < int , int > pii ;
typedef pair < LL , int > pli ;
typedef unsigned long long ULL ;

#define clr( a , x ) memset ( a , x , sizeof a )
#define st first
#define ed second

const int MAXN = 10005 ;
const int BLOCK = 22 ;
```

```
const LL INF = 1e18 ;

namespace Leftist_Tree {
    struct Node {
        int l , r , x , h ;
        LL val ;
    } T[MAXN * 200] ;
    int Root[MAXN] ;
    int node_num ;
    int newnode ( const Node& o ) {
        T[node_num] = o ;
        return node_num ++ ;
    }
    void init () {
        node_num = 1 ;
        T[0].l = T[0].r = T[0].x = T[0].h = 0 ;
        T[0].val = INF ;
    }
    int merge ( int x , int y ) {
        if ( !x ) return y ;
        if ( T[x].val > T[y].val ) swap ( x , y ) ;
        int o = newnode ( T[x] ) ;
        T[o].r = merge ( T[o].r , y ) ;
        if ( T[T[o].l].h < T[T[o].r].h ) swap ( T[o].l , T[o].r ) ;
        T[o].h = T[T[o].r].h + 1 ;
        return o ;
    }
    void insert ( int& x , LL val , int v ) {
        int o = newnode ( T[0] ) ;
        T[o].val = val , T[o].x = v ;
        x = merge ( x , o ) ;
    }
    void show ( int o ) {
        printf ( "%d %lld %lld %lld\n" , o , T[o].val , T[T[o].l].val , T[T[
            o].r].val ) ;
        if ( T[o].l ) show ( T[o].l ) ;
        if ( T[o].r ) show ( T[o].r ) ;
    }
}

using namespace Leftist_Tree ;
vector < pii > G[MAXN] , E[MAXN] ;
int vis[MAXN] ;
int in[MAXN] , p[MAXN] ;
LL d[MAXN] ;
int s , t ;
int n , m , k ;

void addedge ( int u , int v , int c ) {
    G[u].push_back ( pii ( v , c ) ) ;
    E[v].push_back ( pii ( u , c ) ) ;
```

```
}

void dij () {
    priority_queue < pli > q ;
    d[t] = 0 ;
    q.push ( pli ( 0 , t ) ) ;
    while ( !q.empty () ) {
        int u = q.top ().ed ;
        q.pop () ;
        if ( vis[u] ) continue ;
        vis[u] = 1 ;
        for ( int i = 0 ; i < E[u].size () ; ++ i ) {
            int v = E[u][i].st ;
            if ( d[v] > d[u] + E[u][i].ed ) {
                p[v] = u ;
                d[v] = d[u] + E[u][i].ed ;
                q.push ( pli ( -d[v] , v ) ) ;
            }
        }
    }
}

void dfs ( int u ) {
    if ( vis[u] ) return ;
    vis[u] = 1 ;
    if ( p[u] ) Root[u] = Root[p[u]] ;
    int flag = 1 ;
    for ( int i = 0 ; i < G[u].size () ; ++ i ) {
        int v = G[u][i].st ;
        if ( d[v] == INF ) continue ;
        if ( p[u] == v && d[u] == G[u][i].ed + d[v] && flag ) {
            flag = 0 ;
            continue ;
        }
        LL val = d[v] - d[u] + G[u][i].ed ;
        insert ( Root[u] , val , v ) ;
    }
    for ( int i = 0 ; i < E[u].size () ; ++ i ) {
        if ( p[E[u][i].st] == u ) dfs ( E[u][i].st ) ;
    }
}

void solve () {
    for ( int i = 1 ; i <= n ; ++ i ) {
        G[i].clear () ;
        E[i].clear () ;
        d[i] = INF ;
        vis[i] = 0 ;
        p[i] = 0 ;
    }
    for ( int i = 0 ; i < m ; ++ i ) {
```

```
        int u , v , c ;
        scanf ( "%d%d%d" , &u , &v , &c ) ;
        addedge ( u , v , c ) ;
    }
    scanf ( "%d%d%d" , &s , &t , &k ) ;
    dij () ;
    if ( d[s] == INF ) {
        printf ( "-1\n" ) ;
        return ;
    }
    if ( s != t ) -- k ;
    if ( !k ) {
        printf ( "%lld\n" , d[s] ) ;
        return ;
    }
    for ( int i = 1 ; i <= n ; ++ i ) {
        vis[i] = 0 ;
    }
    init () ;
    Root[t] = 0 ;
    dfs ( t ) ;
    priority_queue < pli , vector < pli > , greater < pli > > q ;
    if ( Root[s] ) q.push ( pli ( d[s] + T[Root[s]].val , Root[s] ) ) ;
    while ( k -- ) {
        if ( q.empty () ) {
            printf ( "-1\n" ) ;
            return ;
        }
        pli u = q.top () ;
        q.pop () ;
        if ( !k ) {
            printf ( "%lld\n" , u.st ) ;
            return ;
        }
        int x = T[u.ed].l , y = T[u.ed].r , v = T[u.ed].x ;
        if ( Root[v] ) q.push ( pli ( u.st + T[Root[v]].val , Root[v] ) ) ;
        if ( x ) q.push ( pli ( u.st + T[x].val - T[u.ed].val , x ) ) ;
        if ( y ) q.push ( pli ( u.st + T[y].val - T[u.ed].val , y ) ) ;
    }
}

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

## spfa 费用流

```
/*
```

```
    调用minCostMaxflow(s,t,cost)返回s到t的最大流,cost保存费用
    多组数据调用Ginit()
*/
struct E{
    int v,n,F,f,cost;
}G[M];
int point[N],cnt;
int pre[N];
int dis[N];
bool vis[N];
void Ginit(){
    cnt=1;
    SET(point,0);
}
void addedge(int u,int v,int F,int cost){
    G[++cnt]=(E){v,point[u],F,0,cost},point[u]=cnt;
    G[++cnt]=(E){u,point[v],0,0,-cost},point[v]=cnt;
}
bool spfa(int s,int t){
    queue<int>q;
    SET(vis,0);
    SET(pre,0);
    repab(i,s,t)
        dis[i]=infi;
    dis[s]=0;
    vis[s]=1;
    q.push(s);
    while(!q.empty()){
        int u=q.front();q.pop();
        vis[u]=0;
        for(int i=point[u];i;i=G[i].n){
            int v=G[i].v;
            if(G[i].F>G[i].f&&dis[v]-dis[u]-G[i].cost>0){
                dis[v]=dis[u]+G[i].cost;
                pre[v]=i;
                if(!vis[v]){
                    vis[v]=1;
                    q.push(v);
                }
            }
        }
    }
    return pre[t];
}
int minCostMaxflow(int s,int t,int &cost){
    int f=0;
    cost=0;
    while(spfa(s,t)){
        int Min=infi;
        for(int i=pre[t];i;i=pre[G[i^1].v]){
            if(Min>G[i].F-G[i].f)
```

```
                Min=G[i].F-G[i].f;
        }
        for(int i=pre[t];i;i=pre[G[i^1].v]){
            G[i].f+=Min;
            G[i^1].f-=Min;
            cost+=G[i].cost*Min;
        }
        f+=Min;
    }
    return f;
}
```

## Tarjan 有向图强连通分量

```
/*
    调用SCC()得到强连通分量,调用suodian()缩点
    belong[i]为所在scc编号,sccnum为scc数量
    原图用addedge,存在G,缩点后的图用addedge2,存在G1
    多组数据时调用Ginit()
*/
int n, m;
int point[N], cnt;
int low[N], dfn[N], belong[N], Stack[N];
bool instack[N];
int dfsnow, Stop, sccnum;
struct E{
    int u, v, nex;
}G[M],G1[M];
void tarjan(int u){
    int v;
    dfn[u] = low[u] = ++dfsnow;
    instack[u] = 1;
    Stack[++Stop] = u;
    for (int i = point[u];i;i = G[i].nex){
        v = G[i].v;
        if (!dfn[v]){
            tarjan(v);
            low[u] = min(low[u], low[v]);
        }
        else
            if (instack[v])
                low[u] = min(low[u], dfn[v]);
    }
    if (dfn[u] == low[u]){
        sccnum++;
        do{
            v = Stack[Stop--];
            instack[v] = 0;
            belong[v] = sccnum;
```

```
                num[sccnum][++num[sccnum][0]] = v;
            }
            while (v != u);
        }
}
void Ginit(){
    cnt = 0;
    SET(point,0);
}
void SCC(){
    Stop = sccnum = dfsnow = 0;
    SET(dfn, 0);
    rep(i,n)
        if (!dfn[i])
            tarjan(i);
}
void addedge(int a, int b){
    G[++cnt] = (E){a,b,point[a]}, point[a] = cnt;
}
void addedge2(int a, int b){
    G1[++cnt] = (E){a,b,point[a]}, point[a] = cnt;
}
int degree[N];
void suodian(){
    Ginit();
    SET(degre,0);
    rep(i,m)
        if (belong[G[i].u] != belong[G[i].v]){
            addedge2(belong[G[i].u], belong[G[i].v]);
            degre[belong[G[i].v]]++;
        }
}
/*
    割点和桥
    割点：删除后使图不连通
    桥(割边)：删除后使图不连通
    对图深度优先搜索，定义DFS(u)为u在搜索树（以下简称为树）中被遍历到的次序号。定义Low(u)为u或u的子树中能通过非树边追溯到的DFS序号最小的节点。
       ( )=  {  ( );  ( ),( , )为非树边;  ( ),( , )为树边}
    一个顶点u是割点，当且仅当满足(1)或(2)
    (1) u为树根，且u有多于一个子树。 (2) u不为树根，且满足存在(u,v)为树边，使得DFS(u)<=Low(v)。
    一条无向边(u,v)是桥，当且仅当(u,v)为树边，且满足DFS(u)<Low(v)。
*/
```

## zkw 费用流

```
/*
    调用zkw(s,t,cost)返回s到t的最大流,cost保存费用
```

```
    多组数据调用Ginit()
*/
struct E{
    int v,n,F,f,c;
}G[M];
int point[N],cnt;
int dis[N];
bool vis[N];
void Ginit(){
    cnt=1;
    SET(point,0);
}
void addedge(int u,int v,int F,int cost){
    G[++cnt]=(E){v,point[u],F,0,cost},point[u]=cnt;
    G[++cnt]=(E){u,point[v],0,0,-cost},point[v]=cnt;
}
bool spfa(int s,int t){
    queue<int>q;
    SET(vis,0);
    repab(i,s,t)
        dis[i]=infi;
    dis[s]=0;
    vis[s]=1;
    q.push(s);
    while(!q.empty()){
        int u=q.front();q.pop();
        vis[u]=0;
        for(int i=point[u];i;i=G[i].n){
            int v=G[i].v;
            if(G[i].F>G[i].f&&dis[v]-dis[u]-G[i].c>0){
                dis[v]=dis[u]+G[i].c;
                if(!vis[v]){
                    vis[v]=1;
                    q.push(v);
                }
            }
        }
    }
    return dis[t]!=infi;
}
bool mark[N];
int dfs(int u,int t,int f,int &ans){
    mark[u]=1;
    if(u==t)return f;
    double w;
    int used=0;
    for(int i=point[u];i;i=G[i].n){
        if(G[i].F>G[i].f&&!mark[G[i].v]&&dis[u]+G[i].c-dis[G[i].v]==0){
            w=dfs(G[i].v,t,min(G[i].F-G[i].f,f-used),ans);
            G[i].f+=w;
            G[i^1].f-=w;
```

```
                ans+=G[i].c*w;
                used+=w;
                if(used==f)return f;
            }
        }
        return used;
}
int zkw(int s,int t,int &ans){
        int tmp=0;
        ans=0;
        while(spfa(s,t)){
            mark[t]=1;
            while(mark[t]){
                SET(mark,0);
                tmp+=dfs(s,t,infi,ans);
            }
        }
        return tmp;
}
```

## 倍增 LCA

```
/*
        调用init(),且处理出dep数组后
        调用lca(x,y)得到x,y的lca
*/
int p[M], f[N][M];
void init(){
        p[0] = 1;
        rep(i,M-1){
            p[i] = p[i-1]<<1;
            rep(j,n)
                if(f[j][i-1])
                    f[j][i] = f[f[j][i-1]][i-1]
        }
}
int lca(int x,int y){
        if(dep[x] > dep[y])
            swap(x, y);
        if(dep[x] < dep[y])
            Rep(i,M)
                if((dep[y] - dep[x]) & p[i])
                    y = f[y][i];
        Repr(i,M)
            if(f[x][i] != f[y][i]){
                x = f[x][i];
                y = f[y][i];
            }
        if(x != y)
```

```
        return f[x][0];
    return x;
}
```

## 点分治

```
/*
    问有多少对点它们两者间的距离小于等于K
*/
#include <algorithm>
#include <cstring>
#include <cstdio>
#include <bitset>
#include <queue>
using namespace std;
#define N 40002
int n, K, dis[N], point[N], cnt, siz[N], maxs[N], r, son[N], ans;
bitset<N> vis;
struct E
{
    int v, w, next;
}G[N<<1];
inline void add(int u, int v, int w)
{
    G[++cnt] = (E){v, w, point[u]}, point[u] = cnt;
    G[++cnt] = (E){u, w, point[v]}, point[v] = cnt;
}
inline void getroot(int u, int f)
{
    siz[u] = 1, maxs[u] = 0;
    for (int i = point[u];i;i = G[i].next)
    {
        if (G[i].v == f || vis[G[i].v])continue;
        getroot(G[i].v, u);
        siz[u] += siz[G[i].v];
        maxs[u] = max(maxs[u], siz[G[i].v]);
    }
    maxs[u] = max(maxs[u], n-siz[u]);
    if (maxs[r] > maxs[u])
        r = u;
}
queue<int> Q;
bitset<N> hh;
inline void bfs(int u)
{
    hh.reset();
    Q.push(u);
    hh[u] = 1;
    while (!Q.empty())
```

```
    {
        int i = Q.front();Q.pop();
        for (int p = point[i];p;p = G[p].next)
        {
            if (hh[G[p].v] || vis[G[p].v])continue;
            son[++son[0]] = dis[G[p].v] = dis[i] + G[p].w;
            hh[G[p].v] = 1;
            Q.push(G[p].v);
        }
    }
}
/*inline void dfs(int u, int f)
{
    for (int i = point[u];i;i = G[i].next)
    {
        if (G[i].v == f || vis[G[i].v])continue;
        son[++son[0]] = dis[G[i].v] = dis[u] + G[i].w;
        dfs(G[i].v, u);
    }
}*/
inline int calc(int u)
{
    int res(0), i;
    son[son[0]=1] = dis[u], bfs(u);
    sort(son+1, son+son[0]+1);
    son[++son[0]] = 1<<30;
    for (i = 1;i <= son[0]; ++i)
    {
        if (son[i] > K)continue;
        int x = upper_bound(son+1, son+1+son[0], K-son[i])-(son);
        res += x-1;
        if (son[i] << 1 <= K)res--;
    }
    return res;
}
inline void solve(int u)
{
    dis[u] = 0, vis[u] = 1;
    ans += calc(u);
    for (int i = point[u];i;i = G[i].next)
    {
        if (vis[G[i].v])continue;
        dis[G[i].v] = G[i].w, ans -= calc(G[i].v);
        n = siz[G[i].v];
        maxs[r=0] = N, getroot(G[i].v, 0);
        solve(r);
    }
}
int main()
{
    int i, j, u, v, w;
```

```
    scanf("%d", &n);
    memset(point, 0, sizeof(point));
    vis.reset();
    for (i = 1;i < n; ++i)
    {
        scanf("%d %d %d", &u, &v, &w);
        add(u, v, w);
    }
    scanf("%d", &K);
    maxs[r=0]=n+1;
    getroot(1, 0);
    solve(r);
    printf("%d\n", ans>>1);
    ans = 0;
    return 0;
}
/*
    给一棵树,每条边有权.求一条简单路径,权值和等于K,且边的数量最小
*/
#include <cstdio>
#include <cstring>
#include <bitset>
#include <algorithm>
using namespace std;
#define N 200005
#define Max (N<<1)
bitset<N> vis;
struct hh
{
    int i, x;
    bool operator < (const hh &nb) const
    {
        return x < nb.x;
    }
}son[N];
int n, K, siz[N], maxs[N], dfn[N], point[N], belong[N], dis[N], dep[N], cnt,
     r, ans(Max);
char c;
inline void read(int &x)
{
    for (c = getchar();c > '9' || c < '0';c = getchar());
    for (x = 0;c >= '0' && c <= '9';c = getchar())
        x = (x << 3) + (x << 1) + c - '0';
}
struct E
{
    int v, w, next;
}G[N<<1];
inline void add(int u, int v, int w)
{
    G[++cnt] = (E){v, w, point[u]}, point[u] = cnt;
```

```
    G[++cnt] = (E){u, w, point[v]}, point[v] = cnt;
}
inline void getroot(int u, int f)
{
    siz[u] = 1, maxs[u] = 0;
    for (int i = point[u];i;i = G[i].next)
    {
        int v = G[i].v;
        if (v == f || vis[v])continue;
        getroot(v, u);
        siz[u] += siz[v], maxs[u] = max(maxs[u], siz[v]);
    }
    maxs[u] = max(maxs[u], n-siz[u]);
    if (maxs[u] < maxs[r])r = u;
}
inline void dfs(int u, int f)
{
    if (f != r)belong[u] = belong[f];
    for (int i = point[u];i;i = G[i].next)
    {
        int v = G[i].v;
        if (v == f || vis[v])continue;
        dep[v] = dep[u]+1;
        son[++son[0].i].x = dis[v] = dis[u] + G[i].w;
        son[son[0].i].i = v;
        dfs(v, u);
    }
    dfn[u] = ++cnt;
}
inline int calc(int u)
{
    int res(Max);
    son[++son[0].i].x = dis[u];
    son[1].i = u;
    belong[u] = u;
    for (int i = point[u];i;i = G[i].next)
    {
        int v = G[i].v;
        if (vis[v])continue;
        belong[v] = v;
    }
    dfs(u, 0);
    sort(son+1, son+1+son[0].i);
    son[++son[0].i].x = K<<1;
    for (int i = 1;i <= son[0].i; ++i)
    {
        son[i].x = K - son[i].x;
        int x = lower_bound(son+1, son+1+son[0].i, son[i])-(son);
        for (;son[i].x == son[x].x; ++x)
        {
            if (x == i)continue;
```

```
                if (belong[son[i].i] == belong[son[x].i]) continue;
                res = min(res, dep[son[i].i]-dep[u]+dep[son[x].i]-dep[u]);
            }
            son[i].x = K - son[i].x;
        }
        return res;
}
inline void solve(int u)
{
    son[0].i = dis[u] = 0;
    vis[u] = 1;
    ans = min(ans, calc(u));
    for (int i = point[u]; i; i = G[i].next)
    {
        int v = G[i].v;
        if (vis[v]) continue;
        maxs[r=0] = N-1;
        n = siz[v];
        getroot(v, 0);
        solve(r);
    }
}
int main()
{
//  freopen("a.in", "r", stdin);
    int i, u, v, w;
    read(n), read(K);
//  scanf("%d %d", &n, &K);
    for (i = 1; i < n; ++i)
    {
        read(u), read(v), read(w);
        //scanf("%d %d %d", &u, &v, &w);
        add(u+1, v+1, w);
    }
    maxs[cnt=r=0] = N-1;
    getroot(1, 0);
    solve(r);
    printf("%d\n", ans == Max ? -1 : ans);
}
```

## 堆优化 dijkstra

```
/*
    调用Dijkstra(s)得到从s出发的最短路,存在dist中
    多组数据时调用Ginit()
*/
struct qnode{
    int v,c;
    bool operator <(const qnode &r)const{
```

```
        return c>r.c;
    }
};
struct E{
    int v,w,n;
}G[M];
int point[N], cnt;
bool vis[N];
int dist[N];
void Dijkstra(int s){
    SET(vis,0);
    SET(dist,127);
    dist[s]=0;
    priority_queue<qnode> que;
    while(!que.empty())que.pop();
    que.push((qnode){s,0});
    qnode tmp;
    while(!que.empty()){
        tmp=que.top();
        que.pop();
        int u=tmp.v;
        if(vis[u])continue;
        vis[u]=1;
        for_each_edge(u){
            int v = G[i].v;
            if(!vis[v]&&dist[v]>dist[u]+G[i].w){
                dist[v]=dist[u]+G[i].w;
                que.push((qnode){v,dist[v]});
            }
        }
    }
}
void addedge(int u,int v,int w){
    G[++cnt] = (E){v,w,point[u]}, point[u] = cnt;
}
void Ginit(){
    cnt = 0;
    SET(point,0);
}
```

## 矩阵树定理

```
/*
    矩阵树定理
    令g为度数矩阵,a为邻接矩阵
    生成树的个数为g-a的任何一个n-1阶主子式的行列式的绝对值
    det(a,n)返回n阶矩阵a的行列式
    所以直接调用det(g-a,n-1)就得到答案
    O(n^3)
```

```
    有取模版和double版
    无向图生成树的个数与根无关
    有必选边时压缩边
    有向图以i为根的树形图的数目=基尔霍夫矩阵去掉第i行和第i列的主子式的行列式
        的值(即Matrix-Tree定理不仅适用于求无向图生成树数目,也适用于求有向图
        树形图数目)
*/
int det(int a[N][N], int n){
    rep(i,n)
        rep(j,n)
            a[i][j]=(a[i][j]+mod)%mod;
    ll ans=1,f=1;
    rep(i,n){
        repab(j,i+1,n){
            ll A=a[i][i],B=a[j][i];
            while(B!=0){
                ll t=A/B;A%=B;swap(A,B);
                repab(k,i,n)
                    a[i][k]=(a[i][k]-t*a[j][k]%mod+mod)%mod;
                repab(k,i,n)
                    swap(a[i][k],a[j][k]);
                f=-f;
            }
        }
        if(!a[i][i])return 0;
        ans=ans*a[i][i]%mod;
    }
    if(f==-1)return (mod-ans)%mod;
    return ans;
}
double det(double a[N][N],int n){
    int i, j, k, sign = 0;
    double ret = 1, t;
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            b[i][j] = a[i][j];
    for (i = 1; i <= n; i++) {
        if (zero(b[i][i])) {
            for (j = i + 1; j <= n; j++)
                if (!zero(b[j][i]))
                    break;
            if (j > n)
                return 0;
            for (k = i; k <= n; k++)
                t = b[i][k], b[i][k] = b[j][k], b[j][k] = t;
            sign++;
        }
        ret *= b[i][i];
        for (k = i + 1; k <= n; k++)
            b[i][k] /= b[i][i];
        for (j = i + 1; j <= n; j++)
```

```
            for (k = i + 1; k <= n; k++)
                b[j][k] -= b[j][i] * b[i][k];
    }
    if (sign & 1)
        ret = -ret;
    return ret;
}
/*
    最小生成树计数
*/
#define dinf 1e10
#define linf (LL)1<<60
#define LL long long
#define clr(a,b) memset(a,b,sizeof(a))
LL mod;
struct Edge{
    int a,b,c;
    bool operator<(const Edge & t)const{
        return c<t.c;
    }
}edge[M];
int n,m;
LL ans;
int fa[N],ka[N],vis[N];
LL gk[N][N],tmp[N][N];
vector<int>gra[N];
int findfa(int a,int b[]){return a==b[a]?a:b[a]=findfa(b[a],b);}
LL det(LL a[][N],int n){
    for(int i=0;i<n;i++)for(int j=0;j<n;j++)a[i][j]%=mod;
    long long ret=1;
    for(int i=1;i<n;i++){
        for(int j=i+1;j<n;j++)
            while(a[j][i]){
                LL t=a[i][i]/a[j][i];
                for(int k=i;k<n;k++)
                    a[i][k]=(a[i][k]-a[j][k]*t)%mod;
                for(int k=i;k<n;k++)
                    swap(a[i][k],a[j][k]);
                ret=-ret;
            }
        if(a[i][i]==0)return 0;
        ret=ret*a[i][i]%mod;
        //ret%=mod;
    }
    return (ret+mod)%mod;
}
int main(){
    while(scanf("%d%d%I64d",&n,&m,&mod)==3){
        if(n==0 && m==0 && mod==0)break;
        memset(gk,0,sizeof(gk));
        memset(tmp,0,sizeof(tmp));
```

```
        memset(fa,0,sizeof(fa));
        memset(ka,0,sizeof(ka));
        memset(tmp,0,sizeof(tmp));
        for(int i=0;i<N;i++)gra[i].clear();
        for(int i=0;i<m;i++)
            scanf("%d%d%d",&edge[i].a,&edge[i].b,&edge[i].c);
        sort(edge,edge+m);
        for(int i=1;i<=n;i++)fa[i]=i,vis[i]=0;
        int pre=-1;
        ans=1;
        for(int h=0;h<=m;h++){
            if(edge[h].c!=pre||h==m){
                for(int i=1;i<=n;i++)
                    if(vis[i]){
                        int u=findfa(i,ka);
                        gra[u].push_back(i);
                        vis[i]=0;
                    }
                for(int i=1;i<=n;i++)
                    if(gra[i].size()>1){
                        for(int a=1;a<=n;a++)
                            for(int b=1;b<=n;b++)
                                tmp[a][b]=0;
                        int len=gra[i].size();
                        for(int a=0;a<len;a++)
                            for(int b=a+1;b<len;b++){
                                int la=gra[i][a],lb=gra[i][b];
                                tmp[a][b]=(tmp[b][a]-=gk[la][lb]);
                                tmp[a][a]+=gk[la][lb];tmp[b][b]+=gk[la][lb];
                            }
                        long long ret=(long long)det(tmp,len);
                        ret%=mod;
                        ans=(ans*ret%mod)%mod;
                        for(int a=0;a<len;a++)fa[gra[i][a]]=i;
                    }
                for(int i=1;i<=n;i++){
                    ka[i]=fa[i]=findfa(i,fa);
                    gra[i].clear();
                }
                if(h==m)break;
                pre=edge[h].c;
            }
            int a=edge[h].a,b=edge[h].b;
            int pa=findfa(a,fa),pb=findfa(b,fa);
            if(pa==pb)continue;
            vis[pa]=vis[pb]=1;
            ka[findfa(pa,ka)]=findfa(pb,ka);
            gk[pa][pb]++;gk[pb][pa]++;
        }
        int flag=0;
        for(int i=2;i<=n&&!flag;i++)if(ka[i]!=ka[i-1])flag=1;
```

```
        ans%=mod;
        printf("%I64d\n",flag?0:ans);
    }
    return 0;
}
```

## 平面欧几里得距离最小生成树

```
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<algorithm>
#include<iostream>
#include<fstream>
#include<map>
#include<ctime>
#include<list>
#include<set>
#include<queue>
#include<cmath>
#include<vector>
#include<bitset>
#include<functional>
#define x first
#define y second
#define mp make_pair
#define pb push_back
using namespace std;

typedef long long LL;
typedef double ld;

const int MAX=400000+10;
const int NUM=20;

int n;

struct point
{
    LL x,y;
    int num;
    point(){}
    point(LL a,LL b)
    {
        x=a;
        y=b;
    }
}d[MAX];

```

```
int operator < (const point& a,const point& b)
{
    if(a.x!=b.x)return a.x<b.x;
    else return a.y<b.y;
}

point operator - (const point& a,const point& b)
{
    return point(a.x-b.x,a.y-b.y);
}

LL chaji(const point& s,const point& a,const point& b)
{
    return (a.x-s.x)*(b.y-s.y)-(a.y-s.y)*(b.x-s.x);
}

LL dist(const point& a,const point& b)
{
    return (a.x-b.x)*(a.x-b.x)+(b.y-a.y)*(b.y-a.y);
}

struct point3
{
    LL x,y,z;
    point3(){}
    point3(LL a,LL b,LL c)
    {
        x=a;
        y=b;
        z=c;
    }
    point3(point a)
    {
        x=a.x;
        y=a.y;
        z=x*x+y*y;
    }
};

point3 operator - (const point3 a,const point3& b)
{
    return point3(a.x-b.x,a.y-b.y,a.z-b.z);
}

point3 chaji(const point3& a,const point3& b)
{
    return point3(a.y*b.z-a.z*b.y,-a.x*b.z+a.z*b.x,a.x*b.y-a.y*b.x);
}

LL dianji(const point3& a,const point3& b)
{
```

```
    return a.x*b.x+a.y*b.y+a.z*b.z;
}

LL in_circle(point a,point b,point c,point d)
{
    if(chaji(a,b,c)<0)
        swap(b,c);
    point3 aa(a),bb(b),cc(c),dd(d);
    bb=bb-aa;cc=cc-aa;dd=dd-aa;
    point3 f=chaji(bb,cc);
    return dianji(dd,f);
}

struct Edge
{
    int t;
    list<Edge>::iterator c;
    Edge(){}
    Edge(int v)
    {
        t=v;
    }
};
list<Edge> ne[MAX];

void add(int a,int b)
{
    ne[a].push_front(b);
    ne[b].push_front(a);
    ne[a].begin()->c=ne[b].begin();
    ne[b].begin()->c=ne[a].begin();
}

int sign(LL a)
{
    return a>0?1:(a==0?0:-1);
}

int cross(const point& a,const point& b,const point& c,const point& d)
{
    return sign(chaji(a,c,b))*sign(chaji(a,b,d))>0 && sign(chaji(c,a,d))*
        sign(chaji(c,d,b))>0;
}

void work(int l,int r)
{
    int i,j,nowl=l,nowr=r;
    list<Edge>::iterator it;
    if(l+2>=r)
    {
        for(i=l;i<=r;++i)
```

```
            for(j=i+1;j<=r;++j)
                add(i,j);
        return;
    }
    int mid=(l+r)/2;
    work(l,mid);work(mid+1,r);
    int flag=1;
    for(;flag;)
    {
        flag=0;
        point ll=d[nowl],rr=d[nowr];
        for(it=ne[nowl].begin();it!=ne[nowl].end();++it)
        {
            point t=d[it->t];
            LL s=chaji(rr,ll,t);
            if(s>0 || ( s==0 && dist(rr,t)<dist(rr,ll) ) )
            {
                nowl=it->t;
                flag=1;
                break;
            }
        }
        if(flag)
            continue;
        for(it=ne[nowr].begin();it!=ne[nowr].end();++it)
        {
            point t=d[it->t];
            LL s=chaji(ll,rr,t);
            if(s<0 || (s==0 && dist(ll,rr)>dist(ll,t) ) )
            {
                nowr=it->t;
                flag=1;
                break;
            }
        }
    }
    add(nowl,nowr);
    for(;1;)
    {
        flag=0;
        int best=0,dir=0;
        point ll=d[nowl],rr=d[nowr];
        for(it=ne[nowl].begin();it!=ne[nowl].end();++it)
            if(chaji(ll,rr,d[it->t])>0 && ( best==0 || in_circle(ll,rr,d[
                best],d[it->t])<0 ) )
                best=it->t,dir=-1;
        for(it=ne[nowr].begin();it!=ne[nowr].end();++it)
            if(chaji(rr,d[it->t],ll)>0 && ( best==0 || in_circle(ll,rr,d[
                best],d[it->t])<0 ) )
                best=it->t,dir=1;
        if(!best)break;
```

```
        if(dir==-1)
        {
            for(it=ne[nowl].begin();it!=ne[nowl].end();)
                if(cross(ll,d[it->t],rr,d[best]))
                {
                    list<Edge>::iterator ij=it;
                    ++ij;
                    ne[it->t].erase(it->c);
                    ne[nowl].erase(it);
                    it=ij;
                }
                else ++it;
            nowl=best;
        }
        else if(dir==1)
        {
            for(it=ne[nowr].begin();it!=ne[nowr].end();)
                if(cross(rr,d[it->t],ll,d[best]))
                {
                    list<Edge>::iterator ij=it;
                    ++ij;
                    ne[it->t].erase(it->c);
                    ne[nowl].erase(it);
                    it=ij;
                }
                else ++it;
            nowr=best;
        }
        add(nowl,nowr);
    }
}

struct MstEdge
{
    int x,y;
    LL w;
}e[MAX];
int m;

int operator < (const MstEdge& a,const MstEdge& b)
{
    return a.w<b.w;
}

int fa[MAX];

int findfather(int a)
{
    return fa[a]==a?a:fa[a]=findfather(fa[a]);
}

```

```
int Hash[MAX],p[MAX/4][NUM],deep[MAX],place[MAX];
LL dd[MAX/4][NUM];

vector<int> ne2[MAX];
queue<int> q;

LL getans(int u,int v)
{
    if(deep[u]<deep[v])
        swap(u,v);
    LL ans=0;
    int s=NUM-1;
    while(deep[u]>deep[v])
    {
        while(s && deep[p[u][s]]<deep[v])--s;
        ans=max(dd[u][s],ans);
        u=p[u][s];
    }
    s=NUM-1;
    while(u!=v)
    {
        while(s && p[u][s]==p[v][s])--s;
        ans=max(dd[u][s],ans);
        ans=max(dd[v][s],ans);
        u=p[u][s];
        v=p[v][s];
    }
    return ans;
}

int main()
{
#ifndef ONLINE_JUDGE
    freopen("input.txt","r",stdin);freopen("output.txt","w",stdout);
#endif
    int i,j,u,v;
    scanf("%d",&n);
    for(i=1;i<=n;++i)
    {
        cin>>d[i].x>>d[i].y;
        d[i].num=i;
    }
    sort(d+1,d+n+1);
    for(i=1;i<=n;++i)
        place[d[i].num]=i;
    work(1,n);
    for(i=1;i<=n;++i)
        for(list<Edge>::iterator it=ne[i].begin();it!=ne[i].end();++it)
        {
            if(it->t<i)continue;
            ++m;
```

```
                e[m].x=i;
                e[m].y=it->t;
                e[m].w=dist(d[e[m].x],d[e[m].y]);
            }
        sort(e+1,e+m+1);
        for(i=1;i<=n;++i)
            fa[i]=i;
        for(i=1;i<=m;++i)
            if(findfather(e[i].x)!=findfather(e[i].y))
            {
                fa[findfather(e[i].x)]=findfather(e[i].y);
                ne2[e[i].x].pb(e[i].y);
                ne2[e[i].y].pb(e[i].x);
            }
        q.push(1);
        deep[1]=1;
        Hash[1]=1;
        while(!q.empty())
        {
            u=q.front();q.pop();
            for(i=0;i<(int)ne2[u].size();++i)
            {
                v=ne2[u][i];
                if(!Hash[v])
                {
                    Hash[v]=1;
                    p[v][0]=u;
                    dd[v][0]=dist(d[u],d[v]);
                    deep[v]=deep[u]+1;
                    q.push(v);
                }
            }
        }
        for(i=1;(1<<i)<=n;++i)
            for(j=1;j<=n;++j)
            {
                p[j][i]=p[p[j][i-1]][i-1];
                dd[j][i]=max(dd[j][i-1],dd[p[j][i-1]][i-1]);
            }
        int m;
        scanf("%d",&m);
        while(m--)
        {
            scanf("%d%d",&u,&v);
            printf("%.10lf\n",sqrt((ld)getans(place[u],place[v])));
        }
        return 0;
}
```

## 最大流 Dinic

```
/*
    调用maxflow()返回最大流
    S,T为源汇
    addedge(u,v,f,F)F为反向流量
    多组数据时调用Ginit()
*/
struct E{
    int v, f, F, n;
}G[M];
int point[N], D[N], cnt, S, T;
void Ginit(){
    cnt = 1;
    SET(point,0);
}
void addedge(int u, int v, int f, int F){
    G[++cnt] = (E){v, 0, f, point[u]}, point[u] = cnt;
    G[++cnt] = (E){u, 0, F, point[v]}, point[v] = cnt;
}
queue<int> q;
int BFS(){
    SET(D,0);
    q.push(S);
    D[S] = 1;
    while (!q.empty()){
        int u = q.front();q.pop();
        for_each_edge(u)
            if (G[i].F > G[i].f){
                int v = G[i].v;
                if (!D[v]){
                    D[v] = D[u] + 1;
                    q.push(v);
                }
            }
    }
    return D[T];
}
int Dinic(int u, int F){
    if (u == T) return F;
    int f = 0;
    for_each_edge(u){
        if(F<=f)break;
        int v = G[i].v;
        if (G[i].F > G[i].f && D[v] == D[u] + 1){
            int temp = Dinic(v, min(F - f, G[i].F-G[i].f));
            if (temp == 0)
                D[v] = 0;
            else{
                f += temp;
                G[i].f += temp;
```

```
                G[i^1].f -= temp;
            }
        }
    }
    return f;
}
int maxflow(){
    int f = 0;
    while (BFS())
        f += Dinic(S, infi);
    return f;
}
/*
最大权闭合子图
    在一个有向无环图中,每个点都有一个权值。
    现在需要选择一个子图，满足若一个点被选，其后继所有点也会被选。最大化选出的点权和。
    建图方法：源向所有正权点连容量为权的边，所有负权点向汇点连容量为权的绝对值的边。若原图中存在有向边<u,v>，则从u向v连容量为正无穷的边。答案为所有正权点和 － 最大流
最大权密度子图
    在一个带点权带边权无向图中，选出一个子图，使得该子图的点权和与边权和的比值最大。
    二分答案k，问题转为最大化|V|-k|E|
    确定二元关系：如果一条边连接的两个点都被选择，则将获得该边的权值(可能需要处理负权)
二分图最小点权覆盖集
    点覆盖集：在无向图G=(V,E)中，选出一个点集V′，使得对于任意<u,v>属于E，都有u属于V'或v属于V′，则称V′是无向图G的一个点覆盖集。
    最小点覆盖集：在无向图中，包含点数最少的点覆盖集被称为最小点覆盖集。
    这是一个NPC问题，但在二分图中可以用最大匹配模型快速解决。

    最小点权覆盖集：在最小点覆盖集的基础上每个点均被赋上一个点权。
    建模方法：对二分图进行黑白染色，源点向白点连容量为该点点权的边，黑点向汇点连容量为该点点权的边，对于无向边<u,v>，设u为白点，则从u向v连容量为正无穷的边。最小割即为答案。
二分图最大点权独立集
    点独立集：在无向图G=(V,E)中，选出一个点集V′，使得对于任意u,v属于V',<u,v>不属于E'，则称V′是无向图G的一个点独立集。
    最大点独立集：在无向图中，包含点数最多的点独立集被称为最大点独立集。
    |最大独立集| = |V|-|最大匹配数|
    这是一个NPC问题，但在二分图中可以用最大匹配模型快速解决。
    最大点权独立集：在最大点独立集的基础上每个点均被赋上一个点权。
    建模方法：对二分图进行黑白染色，源点向白点连容量为该点点权的边，黑点向汇点连容量为该点点权的边，对于无向边<u,v>，设u为白点，则从u向v连容量为正无穷的边。所有点权-最小割即为答案。
最小路径覆盖
    在一个DAG中，用尽量少的不相交的简单路径覆盖所有的节点。
    最小路径覆盖数=点数-路径中的边数
    建立一个二分图，把原图中的所有节点分成两份（X集合为i，Y集合为i′），如果原来图中有i->j的有向边，则在二分图中建立i->j′的有向边。最终|最小路
```

```
        径覆盖|=|V|-|最大匹配数|

无源汇可行流
    建图方法:
    首先建立附加源点ss和附加汇点tt,对于原图中的边x->y，若限制为[b,c]，那么连边x->y，流量为c-b,对于原图中的某一个点i，记d(i)为流入这个点的所有边的下界和减去流出这个点的所有边的下界和
    若d(i)>0，那么连边ss->i，流量为d(i),若d(i)<0，那么连边i->tt，流量为-d(i)
    求解方法:
        在新图上跑ss到tt的最大流,若新图满流，那么一定存在一种可行流,此时，原图中每一条边的流量应为新图中对应的边的流量+这条边的流量下界
有源汇可行流
    建图方法:在原图中添加一条边t->s，流量限制为[0,inf],即让源点和汇点也满足流量平衡条件,这样就改造成了无源汇的网络流图,其余方法同上
    求解方法:同 无源汇可行流
有源汇最大流
    建图方法:同有源汇可行流
    求解方法:在新图上跑ss到tt的最大流,若新图满流，那么一定存在一种可行流,记此时sigma f(s,i)=sum1,将t->s这条边拆掉，在新图上跑s到t的最大流,记此时sigma f(s,i)=sum2,最终答案即为sum1+sum2
有源汇最小流
    建图方法:同 无源汇可行流
    求解方法:求ss->tt最大流,连边t->s,inf,求ss->tt最大流,答案即为边t->s,inf的实际流量
有源汇费用流
    建图方法:首先建立附加源点ss和附加汇点tt,对于原图中的边x->y，若限制为[b,c]，费用为cost，那么连边x->y，流量为c-b，费用为cost,对于原图中的某一个点i，记d(i)为流入这个点的所有边的下界和减去流出这个点的所有边的下界和,若d(i)>0，那么连边ss->i，流量为d(i)，费用为0,若d(i)<0，那么连边i->tt，流量为-d(i)，费用为0,连边t->s，流量为inf，费用为0
    求解方法:跑ss->tt的最小费用最大流,答案即为（求出的费用+原图中边的下界*边的费用）
    注意:有上下界的费用流指的是在满足流量限制条件和流量平衡条件的情况下的最小费用流,而不是在满足流量限制条件和流量平衡条件并且满足最大流的情况下的最小费用流,也就是说，有上下界的费用流只需要满足网络流的条件就可以了，而普通的费用流是满足一般条件并且满足是最大流的基础上的最小费用*/
```

## 最大团

```
/*
    用二维bool数组a[][]保存邻接矩阵,下标0~n-1
    建图:Maxclique G = Maxclique(a, n)
    求最大团:mcqdyn(保存最大团中点的数组, 保存最大团中点数的变量)
*/
typedef bool BB[N];
struct Maxclique {
    const BB* e; int pk, level; const float Tlimit;
    struct Vertex{ int i, d; Vertex(int i):i(i),d(0){} };
```

```
typedef vector<Vertex> Vertices; typedef vector<int> ColorClass;
Vertices V; vector<ColorClass> C; ColorClass QMAX, Q;
static bool desc_degree(const Vertex &vi, const Vertex &vj){
    return vi.d > vj.d;
}
void init_colors(Vertices &v){
    const int max_degree = v[0].d;
    for(int i = 0; i < (int)v.size(); i++) v[i].d = min(i, max_degree) + 1;
}
void set_degrees(Vertices &v){
    for(int i = 0, j; i < (int)v.size(); i++)
        for(v[i].d = j = 0; j < int(v.size()); j++)
            v[i].d += e[v[i].i][v[j].i];
}
struct StepCount{ int i1, i2; StepCount():i1(0),i2(0){} };
vector<StepCount> S;
bool cut1(const int pi, const ColorClass &A){
    for(int i = 0; i < (int)A.size(); i++) if (e[pi][A[i]]) return true;
    return false;
}
void cut2(const Vertices &A, Vertices &B){
    for(int i = 0; i < (int)A.size() - 1; i++)
        if(e[A.back().i][A[i].i])
            B.push_back(A[i].i);
}
void color_sort(Vertices &R){
    int j = 0, maxno = 1, min_k = max((int)QMAX.size() - (int)Q.size() + 1, 1);
    C[1].clear(), C[2].clear();
    for(int i = 0; i < (int)R.size(); i++) {
        int pi = R[i].i, k = 1;
        while(cut1(pi, C[k])) k++;
        if(k > maxno) maxno = k, C[maxno + 1].clear();
        C[k].push_back(pi);
        if(k < min_k) R[j++].i = pi;
    }
    if(j > 0) R[j - 1].d = 0;
    for(int k = min_k; k <= maxno; k++)
        for(int i = 0; i < (int)C[k].size(); i++)
            R[j].i = C[k][i], R[j++].d = k;
}
void expand_dyn(Vertices &R){// diff -> diff with no dyn
    S[level].i1 = S[level].i1 + S[level - 1].i1 - S[level].i2;//diff
    S[level].i2 = S[level - 1].i1;//diff
    while((int)R.size()) {
        if((int)Q.size() + R.back().d > (int)QMAX.size()){
            Q.push_back(R.back().i); Vertices Rp; cut2(R, Rp);
            if((int)Rp.size()){
                if((float)S[level].i1 / ++pk < Tlimit) degree_sort(Rp); //diff
```

```
                    color_sort(Rp);
                    S[level].i1++, level++;//diff
                    expand_dyn(Rp);
                    level--;//diff
                }
                else if((int)Q.size() > (int)QMAX.size()) QMAX = Q;
                Q.pop_back();
            }
            else return;
            R.pop_back();
        }
    }
    void mcqdyn(int* maxclique, int &sz){
        set_degrees(V); sort(V.begin(),V.end(), desc_degree); init_colors(V)
            ;
        for(int i = 0; i < (int)V.size() + 1; i++) S[i].i1 = S[i].i2 = 0;
        expand_dyn(V); sz = (int)QMAX.size();
        for(int i = 0; i < (int)QMAX.size(); i++) maxclique[i] = QMAX[i];
    }
    void degree_sort(Vertices &R){
        set_degrees(R); sort(R.begin(), R.end(), desc_degree);
    }
    Maxclique(const BB* conn, const int sz, const float tt = 0.025) \
     : pk(0), level(1), Tlimit(tt){
        for(int i = 0; i < sz; i++) V.push_back(Vertex(i));
        e = conn, C.resize(sz + 1), S.resize(sz + 1);
    }
};
```

## 最小度限制生成树

```
/*
    只限制一个点的度数
*/
#include <iostream>
#include <cstdio>
#include <cmath>
#include <vector>
#include <cstring>
#include <algorithm>
#include <string>
#include <set>
#include <ctime>
#include <queue>
#include <map>

#define CL(arr, val)    memset(arr, val, sizeof(arr))
#define REP(i, n)       for((i) = 0; (i) < (n); ++(i))
#define FOR(i, l, h)    for((i) = (l); (i) <= (h); ++(i))
```

```
#define FORD(i, h, l)   for((i) = (h); (i) >= (l); --(i))
#define L(x)     (x) << 1
#define R(x)     (x) << 1 | 1
#define MID(l, r)    (l + r) >> 1
#define Min(x, y)    x < y ? x : y
#define Max(x, y)    x < y ? y : x
#define E(x)     (1 << (x))

const double eps = 1e-8;
typedef long long LL;
using namespace std;
const int inf = ~0u>>2;
const int N = 33;

int parent[N];
int g[N][N];
bool flag[N][N];
map<string, int> NUM;

int n, k, cnt, ans;

struct node {
    int x;
    int y;
    int v;
} a[1<<10];

struct edge {
    int x;
    int y;
    int v;
} dp[N];

bool cmp(node a, node b) {
    return a.v < b.v;
}

int find(int x) {   //并查集查找
    int k, j, r;
    r = x;
    while(r != parent[r]) r = parent[r];
    k = x;
    while(k != r) {
        j = parent[k];
        parent[k] = r;
        k = j;
    }
    return r;
}

int get_num(string s) {    //求编号
```

```
    if(NUM.find(s) == NUM.end()) {
        NUM[s] = ++cnt;
    }
    return NUM[s];
}

void kruskal() {  //。。。
    int i;
    FOR(i, 1, n) {
        if(a[i].x == 1 || a[i].y == 1)  continue;
        int x = find(a[i].x);
        int y = find(a[i].y);
        if(x == y)  continue;
        flag[a[i].x][a[i].y] = flag[a[i].y][a[i].x] = true;
        parent[y] = x;
        ans += a[i].v;
    }
    //printf("%d\n", ans);
}

void dfs(int x, int pre) {   //dfs求1到某节点路程上的最大值
    int i;
    FOR(i, 2, cnt) {
        if(i != pre && flag[x][i]) {
            if(dp[i].v == -1) {
                if(dp[x].v > g[x][i])   dp[i] = dp[x];
                else {
                    dp[i].v = g[x][i];
                    dp[i].x = x;     //记录这条边
                    dp[i].y = i;
                }
            }
            dfs(i, x);
        }
    }
}

void init() {
    ans = 0; cnt = 1;
    CL(flag, false);
    CL(g, -1);
    NUM["Park"] = 1;
    for(int i = 0; i < N; ++i)  parent[i] = i;
}

int main() {
    //freopen("data.in", "r", stdin);

    int i, j, v;
    string s;
    scanf("%d", &n);
```

```
    init();
    for(i = 1; i <= n; ++i) {
        cin >> s;
        a[i].x = get_num(s);
        cin >> s;
        a[i].y = get_num(s);
        scanf("%d", &v);
        a[i].v = v;
        if(g[a[i].x][a[i].y] == -1)     g[a[i].x][a[i].y] = g[a[i].y][a[i].x] = v;
        else     g[a[i].x][a[i].y] = g[a[i].y][a[i].x] = min(g[a[i].x][a[i].y], v);
    }
    scanf("%d", &k);
    int set[N], Min[N];
    REP(i, N)    Min[i] = inf;
    sort(a + 1, a + n + 1, cmp);
    kruskal();
    FOR(i, 2, cnt) {      //找到1到其他连通块的最小值
        if(g[1][i] != -1) {
            int x = find(i);
            if(Min[x] > g[1][i]) {
                Min[x] = g[1][i];
                set[x] = i;
            }
        }
    }
    int m = 0;
    FOR(i, 1, cnt) {   //把1跟这些连通块连接起来
        if(Min[i] != inf) {
            m++;
            flag[1][set[i]] = flag[set[i]][1] = true;
            ans += g[1][set[i]];
        }
    }
    //printf("%d\n", ans);
    for(i = m + 1; i <= k; ++i) {   //从度为m+1一直枚举到最大为k，找ans的最小值
        CL(dp, -1);
        dp[1].v = -inf;     //dp初始化
        for(j = 2; j <= cnt; ++j) {
            if(flag[1][j])   dp[j].v = -inf;
        }
        dfs(1, -1);
        int tmp, mi = inf;
        for(j = 2; j <= cnt; ++j) {
            if(g[1][j] != -1) {
                if(mi > g[1][j] - dp[j].v) {      //找到一条dp到连通块中某个点的边，替换原来连通块中的边（前提是新找的这条边比原来连通块中那条边要大）
                    mi = g[1][j] - dp[j].v;
```

```
                    tmp = j;
                }
            }
        }
        if(mi >= 0) break;     //如果不存在这样的边，直接退出
        int x = dp[tmp].x, y = dp[tmp].y;

        flag[1][tmp] = flag[tmp][1] = true;    //加上新找的边
        flag[x][y] = flag[y][x] = false;     //删掉被替换掉的那条边

        ans += mi;
    }
    printf("Total miles driven: %d\n", ans);

    return 0;

}
```

## 最优比率生成树

```
#include<map>
#include<cmath>
#include<ctime>
#include<queue>
#include<cstdio>
#include<vector>
#include<bitset>
#include<cstring>
#include<iostream>
#include<algorithm>
#define ll long long
#define mod 1000000009
#define inf 1000000000
#define eps 1e-8
using namespace std;
int n,cnt;
int x[1005],y[1005],z[1005],last[1005];
double d[1005],mp[1005][1005],ans;
bool vis[1005];
void prim(){
    for(int i=1;i<=n;i++){
        d[i]=inf;vis[i]=0;
    }
    d[1]=0;
    for(int i=1;i<=n;i++){
        int now=0;d[now]=inf;
        for(int j=1;j<=n;j++)if(d[j]<d[now]&&!vis[j])now=j;
        ans+=d[now];vis[now]=1;
        for(int j=1;j<=n;j++)
```

```
                if(mp[now][j]<d[j]&&!vis[j])
                    d[j]=mp[now][j];
    }
}
double sqr(double x){
    return x*x;
}
double dis(int a,int b){
    return sqrt(sqr(x[a]-x[b])+sqr(y[a]-y[b]));
}
void cal(double mid){
    ans=0;
    for(int i=1;i<=n;i++)
        for(int j=i+1;j<=n;j++)
            mp[i][j]=mp[j][i]=abs(z[i]-z[j])-mid*dis(i,j);
    prim();
}
int main(){
    while(scanf("%d",&n)){
        if(n==0)break;
        for(int i=1;i<=n;i++)
            scanf("%d%d%d",&x[i],&y[i],&z[i]);
        double l=0,r=1000;
        for(int i=1;i<=30;i++)
        {
            double mid=(l+r)/2;
            cal(mid);
            if(ans<0)r=mid;
            else l=mid;
        }
        printf("%.3f\n",l);
    }
    return 0;
}
```

# 数学

## 常用公式

## 积性函数

$\sigma_k(n) = \Sigma_{d|n} d^k$ 表示 $n$ 的约数的 $k$ 次幂和

$\sigma_k(n) = \Pi_{i=1}^{num} \frac{(p_i^{a_i+1})^k - 1}{p_i^k - 1}$

$\varphi(n) = \Sigma_{i=1}^{n}[(n,i)=1] = \Pi_{i=1}^{k}(1-\frac{1}{p_i})$

$\varphi(p^k) = (p-1)p^{k-1}$

$\Sigma_{d|n}\varphi(n) = n \rightarrow \varphi(n) = n - \Sigma_{d|n,d<n}$

$n \geq 2$ 时 $\varphi(n)$ 为偶数

$$\mu(n) = \begin{cases} 0 & \text{有平方因子} \\ (-1)^t & n = \Pi_{i=1}^{t} p_i \end{cases}$$

$[n=1] = \Sigma_{d|n}\mu(d)$ 排列组合后二项式定理转换即可证明

$n = \Sigma_{d|n}\varphi(d)$ 将 $\frac{i}{n}(1 \leq i \leq n)$ 化为最简分数统计个数即可证明

## 莫比乌斯反演

$$F(n) = \sum_{d|n} f(d) \Rightarrow f(n) = \sum_{d|n} \mu(d) * F(\tfrac{n}{d})$$
$$F(n) = \sum_{n|d} f(d) \Rightarrow f(n) = \sum_{n|d} \mu(\tfrac{n}{d}) * F(d)$$
$$f(n) = \sum_{d|n} \phi(d) \Rightarrow \phi(n) = \sum_{d|n} \mu(d) f(\tfrac{n}{d}) = \sum_{d|n} \mu(d) \tfrac{n}{d}$$

## 常用等式

不知道有什么用

$\sum_{d|N} \phi(d) = N$

$\sum_{i \leq N} i * [(i,N) = 1] = \frac{N*\phi(N)}{2}$

$\sum_{d|N} \frac{\mu(d)}{d} = \frac{\phi(N)}{N}$

常用代换

$\sum_{d|N} \mu(d) = [N=1]$

考虑每个数的贡献

$\sum_{i \leq N} \lfloor \frac{N}{i} \rfloor = \sum_{i \leq N} d(i)$

## SG 函数

```
#define MAX 150 //最大的步数

int step[MAX], sg[10500], steps;    //使用前应将sg初始化为-1
//step:所有可能的步数，要求从小到大排序
//steps:step的大小
//sg:存储sg的值


int getsg(int m)
{
    int hashs[MAX] = {0};
    int i;
    for (i = 0; i < steps; i++)
    {
        if (m - step[i] < 0) {
            break;
        }
        if (sg[m - step[i]] == -1) {
```

```
            sg[m - step[i]] = getsg(m - step[i]);
        }
        hashs[sg[m - step[i]]] = 1;
    }
    for (i = 0;; i++) {
        if (hashs[i] == 0) {
            return i;
        }
    }
}

/*
Array(存储可以走的步数，Array[0]表示可以有多少种走法)
Array[]需要从小到大排序
1.可选步数为1~m的连续整数，直接取模即可，SG(x)=x%(m+1);
2.可选步数为任意步，SG(x) = x;
3.可选步数为一系列不连续的数，用GetSG(计算)
*/
//获取sg表
int SG[MAX], hashs[MAX];

void init(int Array[], int n)
{
    int i, j;
    memset(SG, 0, sizeof(SG));
    for (i = 0; i <= n; i++)
    {
        memset(hashs, 0, sizeof(hashs));
        for (j = 1; j <= Array[0]; j++)
        {
            if (i < Array[j]) {
                break;
            }
            hashs[SG[i - Array[j]]] = 1;
        }
        for (j = 0; j <= n; j++)
        {
            if (hashs[j] == 0)
            {
                SG[i] = j;
                break;
            }
        }
    }
}
```

## 矩阵乘法快速幂

```
/*
```

```
    MATN为矩阵大小
    MOD为模数
    调用pamt(a,k)返回a^k
*/
struct mat{
    int n, m;
    int c[MATN][MATN];
};
mat cheng(const mat &a, const mat &b){
    mat w;
    SET(w.c,0);
    w.n = a.n, w.m = b.m;
    Rep(i,a.n)Rep(j,b.m)Rep(k,a.m){
        w.c[i][j] += (ll)a.c[i][k] * b.c[k][j] % MOD;
        if(w.c[i][j]>MOD)w.c[i][j]-=MOD;
    }
    return w;
}
mat pmat(mat a, ll k){
    mat i;
    i.n = i.m = MATN;
    SET(i.c,0);
    Rep(i,MATN)
        i.c[i][i] = 1;
    while(k){
        if(k&1)
            i=cheng(i,a);
        a=cheng(a,a);
        k>>=1;
    }
    return i;
}
```

## 线性基

```
/*
    求一条从1到n的路径,使得路径上的边的异或和最大。
*/
#include <cstdio>
#include <algorithm>
using namespace std;
#define N 50001
#define M 100001
struct E
{
    int u, v, next;
    long long w;
    E(int _u = 0, int _v = 0, int _next = 0, long long _w = 0){u = _u, v =
        _v, next = _next, w = _w;}
```

```
}G[M<<1];
int cnt, point[N], n, m;
char c;
template<class T>
inline void read(T &x)
{
    T opt(1);
    for (c = getchar();c > '9' || c < '0';c = getchar())if (c == '-')opt =
        -1;
    for (x = 0;c >= '0' && c <= '9';c = getchar())x = (x << 3) + (x << 1) +
        c - '0';
    x *= opt;
}
bool vis[N];
long long dis[N];
long long a[M<<1];
int Gauss()
{
    int i, j(0), k;
    for (i = 63;i >= 0; --i)
    {
        for (k = j+1;k <= n; ++k)
        if ((a[k] >> i) & 1)break;
        if (k > n)continue;
        swap(a[k], a[j+1]);
        for (k = 1;k <= n; ++k)
            if (j+1 != k && ((a[k] >> i) & 1))
                a[k] ^= a[j+1];
        j++;
    }
    return j;
}inline void dfs(int u)
{
    vis[u] = 1;
    int i, v;
    for (i = point[u];i;i = G[i].next)
    {
        v = G[i].v;
        if (vis[v])
            a[++m] = dis[u] ^ dis[v] ^ G[i].w;
        else
        {
            dis[v] = dis[u] ^ G[i].w;
            dfs(v);
        }
    }
}
int main()
{
    read(n), read(m);
    int i, j, u, v, k;
```

```
    long long w, ans;
    for (i = 1;i <= m; ++i)
    {
        read(u), read(v), read(w);
        G[++cnt] = E(u, v, point[u], w), point[u] = cnt;
        G[++cnt] = E(v, u, point[v], w), point[v] = cnt;
    }
    m = 0;
    dfs(1);
    ans = dis[n];
    n = m;
    k = Gauss();
    for (i = k;i; --i)
        ans = max(ans, ans ^ a[i]);
    printf("%lld\n", ans);
    return 0;
}
```

## 线性筛

```
/*
    is是不是质数
    phi欧拉函数
    mu莫比乌斯函数
    minp最小质因子
    mina最小质因子次数
    d约数个数
*/
int prime[N];
int size;
int is[N];
int phi[N];//欧拉函数
int mu[N];//莫比乌斯函数
int minp[N];//最小质因子
int mina[N];//最小质因子次数
int d[N];//约数个数
void getprime(int list){
    SET(is,1);
    mu[1] = 1;
    phi[1] = 1;
    is[1] = 0;
    repab(i,2,list){
        if(is[i]){
            prime[++size] = i;
            phi[i] = i-1;
            mu[i] = -1;
            minp[i] = i;
            mina[i] = 1;
            d[i] = 2;
```

```
        }
        rep(j,size){
            if(i*prime[j]>list)
                break;
            is[i * prime[j]] = 0;
            minp[i*prime[j]] = prime[j];
            if(i % prime[j] == 0){
                mu[i*prime[j]] = 0;
                phi[i*prime[j]] = phi[i] * prime[j];
                mina[i*prime[j]] = mina[i]+1;
                d[i*prime[j]] = d[i]/(mina[i]+1)*(mina[i]+2);
                break;
            }else{
                phi[i*prime[j]] = phi[i] * (prime[j] - 1);
                mu[i*prime[j]] = -mu[i];
                mina[i*prime[j]] = 1;
                d[i*prime[j]] = d[i]*d[prime[j]];
            }
        }
    }
}
```

## 整数卷积 NTT

```
/*
    计算形式为a[n] = sigma(b[n-i]*c[i])的卷积,结果存在c中
    下标从0开始
    调用juanji(n,b,c)
    P为模数
    G是P的原根
*/
const ll P=998244353;
const ll G=3;
void change(ll y[],int n){
    int b=n>>1,s=n-1;
    for(int i=1,j=n>>1;i<s;i++){
        if(i<j)swap(y[i],y[j]);
        int k=b;
        while(j>=k){
            j-=k;
            k>>=1;
        }
        j+=k;
    }
}
void NTT_(ll y[],int len,int on){
    change(y,len);
    for(int h=2;h<=len;h<<=1){
        ll wn=powm(G,(P-1)/h,P);
```

```
        if(on<0)wh=powm(wh,P-2,P);
        for(int i=0;i<len;i+=h){
            ll w=1;
            int r=h>>1;
            for(int k=i,s=r+i;k<s;k++){
                ll u=y[k];
                ll t=w*y[k+r]%P;
                y[k]=u+t;
                if(y[k]>=P)y[k]-=P;
                y[k+r]=u-t;
                if(y[k+r]<0)y[k+r]+=P;
                w=w*wh%P;
            }
        }
    }
    if(on<0){
        ll I=powm((ll)len,P-2,P);
        Rep(i,len)y[i]=y[i]*I%P;
    }
}
void juanji(int n, ll *b, ll *c){
    int len=1;
    while(len<(n<<1))len<<=1;
    Repab(i,n,len)c[i]= b[i] = 0;
    NTT_(b,len,1);
    NTT_(c,len,1);
    Rep(i,len)
        c[i]= c[i]*b[i]%P;
    NTT_(c,len,-1);
}
```

## 中国剩余定理

```
/*
    合并ai在模mi下的结果为模m_0*m_1*...*m_n-1
*/
inline int exgcd(int a, int b, int &x, int &y){
    if(!b){
        x = 1, y = 0;
        return a;
    }
    else{
        int d = exgcd(b, a % b, x, y), t = x;
        x = y, y = t - a / b * y;
        return d;
    }
}
inline int inv(int a, int p){
    int d, x, y;
```

```
    d = exgcd(a, p, x, y);
    return d == 1 ? (x + p) % p : -1;
}
int china(int n,int *a,int *m){
    int __M = MOD - 1, d, x = 0, y;
    for(int i = 0;i < n; ++i){
        int w = __M / m[i];
        d = exgcd(m[i], w, d, y);
        x = (x + ((long long)y*w%__M)*(long long)a[i]%__M)%__M;
    }
    while(x <= 0)
        x += __M;
    return x;
}
```

# 字符串

## AC 自动机

```
/// AC自动机.

/// mxn: 自动机的节点池子大小.
const int mxn = 105000;

/// ct: 字符集大小.
const int cst = 26;

/// 重新初始化:
node*pt = pool;

////////////////////////////////////////////////////////////////////////////

struct node
{
    node*s[cst];      // Trie 转移边.
    node*trans[cst]; // 自动机转移边.
    node*f;           // Fail 指针.
    char v;           // 当前节点代表字符(父节点指向自己的边代表的字符).
    bool leaf;        // 是否是某个字符串的终点. 注意该值为true不一定是叶子.
    node() { } // 保留初始化.
}
pool[mxn]; node*pt=pool;
node* newnode() { memset(pt, 0, sizeof(node)); return pt++; }

/// 递推队列.
node*qc[mxn];
node*qf[mxn];
int qh,qt;
```

```
struct Trie
{
    node*root;
    Trie(){ root = newnode(); root->v = '*' - 'a'; }

    /// g: 需要插入的字符串; len:长度.
    void Insert(char* g, int len)
    {
        node*x=root;
        for(int i=0;i<len;i++)
        {
            int v = g[i]-'a';
            if(x->s[v] == NULL)
            {
                x->s[v] = newnode();
                x->s[v]->v = v;
            }
            x = x->s[v];
        }
        x->leaf = true;
    }

    /// 在所有字符串插入之后执行.
    /// BFS递推, qc[i]表示队中节点指针, qf表示队中对应节点的fail指针.
    void Construct()
    {
        node*x = root;
        qh = qt = 0;
        for(int i=0; i<cst; i++) if(x->s[i])
        {
            x->s[i]->f = root;
            for(int j=0; j<cst; j++) if(x->s[i]->s[j])
            { qc[qt] = x->s[i]->s[j]; qf[qt]=root; qt++; }
        }

        while(qh != qt)
        {
            node*cur = qc[qh];
            node*fp = qf[qh];
            qh++;

            while(fp != root && fp->s[cur->v] == NULL) fp = fp->f;
            if(fp->s[cur->v]) fp = fp->s[cur->v];
            cur->f = fp;

            for(int i=0; i<cst; i++)
                if(cur->s[i]) { qc[qt] = cur->s[i]; qf[qt] = fp; qt++; }
        }
    }

```

```
    // 拿到转移点.
    // 暴力判定.
    node* GetTrans(node*x, int v)
    {
        while(x != root && x->s[v] == NULL) x = x->f;
        if(x->s[v]) x = x->s[v];
        return x;
    }

    // 拿到转移点.
    // 记忆化搜索.
    node* GetTrans(node*x, int v)
    {
        if(x->s[v]) return x->trans[v] = x->s[v];

        if(x->trans[v] == NULL)
        {
            if(x == root) return root;
            return x->trans[v] = GetTrans(x->f, v);
        }

        return x->trans[v];
    }
};
```

## KMP

```
//KMP算法
//查找成功则返回所在位置(int),否则返回-1.

#define MAXM 100000000  //字符串最大长度

void getNext(char *p, char *next)
{
    int j = 0;
    int k = -1;
    next[0] = -1;
    while (j < n)
    {
        if (k == -1 || p[j] == p[k])
        {
            j++;
            k++;
            next[j] = k;
        }
        else
            k = next[k];
    }
}
```

```
int KMP(char *s, char *p,int m,int n)     //查找成功则返回所在位置(int),否则返回-1.
{                                         //s为文本串,p为模式串;m为文本串长度,n为模式串长度.
    char next[MAXM];
    int i = 0;
    int j = 0;
    getNext(p, next);
    while (i < m)
    {
        if (j == -1 || s[i] == p[j])
        {
            i++;
            j++;
        }
        else
            j = next[j];
        if (j == n)
            return i - n + 1;
    }
    return -1;
}
```

## Manacher

```
#define MAXM 20001
//返回回文串的最大值
//MAXM至少应为输入字符串长度的两倍+1

int p[MAXM];
char s[MAXM];

int manacher(string str) {
    memset(p, 0, sizeof(p));
    int len = str.size();
    int k;
    for (k = 0; k < len; k++) {
        s[2 * k] = '#';
        s[2 * k + 1] = str[k];
    }
    s[2 * k] = '#';
    s[2 * k + 1] = '\0';
    len = strlen(s);
    int mx = 0;
    int id = 0;
    for (int i = 0; i < len; ++i) {
        if ( i < mx ) {
            p[i] = min(p[2 * id - i], mx - i);
```

```
        }
        else {
            p[i] = 1;
        }
        for (; s[i - p[i]] == s[i + p[i]] && s[i - p[i]] != '\0' && s[i + p[i]] != '\0' ; ) {
            p[i]++;
        }
        if ( p[i] + i > mx ) {
            mx = p[i] + i;
            id = i;
        }
    }
    int res = 0;
    for (int i = 0; i < len; ++i) {
        res = max(res, p[i]);
    }
    return res - 1;
}
```

## Trie 树

```
#define CHAR_SIZE 26          //字符种类数
#define MAX_NODE_SIZE 10000   //最大节点数

inline int getCharID(char a) {  //返回a在子数组中的编号
    return a - 'a';
}

struct Trie
{
    int num;//记录多少单词途径该节点，即多少单词拥有以该节点为末尾的前缀
    bool terminal;//若terminal==true，该节点没有后续节点
    int count;//记录单词的出现次数，此节点即一个完整单词的末尾字母
    struct Trie *son[CHAR_SIZE];//后续节点
};

struct Trie trie_arr[MAX_NODE_SIZE];
int trie_arr_point=0;

Trie *NewTrie()
{
    Trie *temp=&trie_arr[trie_arr_point++];
    temp->num=1;
    temp->terminal=false;
    temp->count=0;
    for(int i=0;i<sonnum;++i)temp->son[i]=NULL;
    return temp;
}
```

```
//插入新词,root:树根,s:新词,len:新词长度
void Insert(Trie *root,char *s,int len)
{
    Trie *temp=root;
    for(int i=0;i<len;++i)
        {
            if(temp->son[getCharID(s[i])]==NULL)temp->son[getCharID(s[i])]=
                NewTrie();
            else {temp->son[getCharID(s[i])]->num++;temp->terminal=false;}
            temp=temp->son[getCharID(s[i])];
        }
    temp->terminal=true;
    temp->count++;
}
//删除整棵树
void Delete()
{
    memset(trie_arr,0,trie_arr_point*sizeof(Trie));
    trie_arr_point=0;
}
//查找单词在字典树中的末尾节点.root:树根,s:单词,len:单词长度
Trie* Find(Trie *root,char *s,int len)
{
    Trie *temp=root;
    for(int i=0;i<len;++i)
    if(temp->son[getCharID(s[i])]!=NULL)temp=temp->son[getCharID(s[i])];
    else return NULL;
    return temp;
}
```

## 后缀数组-DC3

```
//dc3函数:s为输入的字符串,sa为结果数组,slen为s长度,m为字符串中字符的最大值+1
//s及sa数组的大小应为字符串大小的3倍.

#define MAXN 100000   //字符串长度

#define F(x) ((x)/3+((x)%3==1?0:tb))
#define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)

int wa[MAXN], wb[MAXN], wv[MAXN], ws[MAXN];

int c0(int *s, int a, int b)
{
    return s[a] == s[b] && s[a + 1] == s[b + 1] && s[a + 2] == s[b + 2];
}

int c12(int k, int *s, int a, int b)
```

```
{
    if (k == 2) return s[a] < s[b] || s[a] == s[b] && c12(1, s, a + 1, b +
        1);
    else return s[a] < s[b] || s[a] == s[b] && wv[a + 1] < wv[b + 1];
}

void sort(int *s, int *a, int *b, int slen, int m)
{
    int i;
    for (i = 0; i < slen; i++) wv[i] = s[a[i]];
    for (i = 0; i < m; i++) ws[i] = 0;
    for (i = 0; i < slen; i++) ws[wv[i]]++;
    for (i = 1; i < m; i++) ws[i] += ws[i - 1];
    for (i = slen - 1; i >= 0; i--) b[--ws[wv[i]]] = a[i];
    return;
}

void dc3(int *s, int *sa, int slen, int m)
{
    int i, j, *rn = s + slen, *san = sa + slen, ta = 0, tb = (slen + 1) / 3,
         tbc = 0, p;
    s[slen] = s[slen + 1] = 0;
    for (i = 0; i < slen; i++) if (i % 3 != 0) wa[tbc++] = i;
    sort(s + 2, wa, wb, tbc, m);
    sort(s + 1, wb, wa, tbc, m);
    sort(s, wa, wb, tbc, m);
    for (p = 1, rn[F(wb[0])] = 0, i = 1; i < tbc; i++)
        rn[F(wb[i])] = c0(s, wb[i - 1], wb[i]) ? p - 1 : p++;
    if (p < tbc) dc3(rn, san, tbc, p);
    else for (i = 0; i < tbc; i++) san[rn[i]] = i;
    for (i = 0; i < tbc; i++) if (san[i] < tb) wb[ta++] = san[i] * 3;
    if (slen % 3 == 1) wb[ta++] = slen - 1;
    sort(s, wb, wa, ta, m);
    for (i = 0; i < tbc; i++) wv[wb[i] = G(san[i])] = i;
    for (i = 0, j = 0, p = 0; i < ta && j < tbc; p++)
        sa[p] = c12(wb[j] % 3, s, wa[i], wb[j]) ? wa[i++] : wb[j++];
    for (; i < ta; p++) sa[p] = wa[i++];
    for (; j < tbc; p++) sa[p] = wb[j++];
    return;
}
```

## 后缀数组-倍增法

```
#define MAXN 100000        //字符串长度

//da函数:s为输入的字符串,sa为结果数组,slen为s长度,m为字符串中字符的最大值+1
//调用前应将s[slen]设为0,因此调用时slen为s长度+1

//calHeight函数:返回sa中排名相邻的两个后缀的最长公共前缀
```

```

int cmp(int *s, int a, int b, int l) {
    return (s[a] == s[b]) && (s[a + l] == s[b + l]);
}

int wa[MAXN], wb[MAXN], ws[MAXN], wv[MAXN];
void da(int *s, int *sa, int slen, int m) {
    int i, j, p, *x = wa, *y = wb, *t;
    for (i = 0; i < m; i++) ws[i] = 0;
    for (i = 0; i < slen; i++) ws[x[i] = s[i]]++;
    for (i = 1; i < m; i++) ws[i] += ws[i - 1];
    for (i = slen - 1; i >= 0; i--) sa[--ws[x[i]]] = i;
    for (j = 1, p = 1; p < slen; j *= 2, m = p)
    {
        for (p = 0, i = slen - j; i < slen; i++) y[p++] = i;
        for (i = 0; i < slen; i++) if (sa[i] >= j) y[p++] = sa[i] - j;
        for (i = 0; i < slen; i++) wv[i] = x[y[i]];
        for (i = 0; i < m; i++) ws[i] = 0;
        for (i = 0; i < slen; i++) ws[wv[i]]++;
        for (i = 1; i < m; i++) ws[i] += ws[i - 1];
        for (i = slen - 1; i >= 0; i--) sa[--ws[wv[i]]] = y[i];
        for (t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < slen; i++)
            x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
    }
}

//----------------------------------------------------------------
int rank[MAXN], height[MAXN];
void calHeight(int *s, int *sa, int slen) {
    int i, j, k = 0;
    for (i = 1; i <= slen; i++) rank[sa[i]] = i;
    for (i = 0; i < slen; height[rank[i++]] = k )
        for (k ? k-- : 0, j = sa[rank[i] - 1]; s[i + k] == s[j + k]; k++);
}
//----------------------------------------------------------------
```

## 后缀自动机

```
/*
    求多个串的LCS
*/
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
#define N 100001
struct node
{
    node *suf, *s[26], *next;
```

```
    int val, w[11];
}*r, *l, T[N<<1+1];
node *point[N];
char str[N];
int n, len, k, tot;
inline void add(int w)
{
    node *p = l, *np = &T[tot++];
    np->val = p->val+1;
    np->next = point[np->val], point[np->val] = np;
    while (p && !p->s[w])
        p->s[w] = np, p = p->suf;
    if (!p)
        np->suf = r;
    else
    {
        node *q = p->s[w];
        if (p->val+1 == q->val)
            np->suf = q;
        else
        {
            node *nq = &T[tot++];
            memcpy(nq->s, q->s, sizeof q->s);
            nq->val = p->val+1;
            nq->next = point[p->val+1], point[p->val+1] = nq;
            nq->suf = q->suf;
            q->suf = nq;
            np->suf = nq;
            while (p && p->s[w] == q)
                p->s[w] = nq, p = p->suf;
        }
    }
    l = np;
}
int main()
{
    freopen("a.in", "r", stdin);
    int i, j, now, L, res, ans(0), w;
    node *p;
    r = l = &T[tot++];
    r->next = point[0], point[0] = r;
    scanf("%s", str);
    L = strlen(str);
    for (i = 0;i < L; ++i)
        add(str[i]-'a');
    for (tot = 1;scanf("%s", str) != EOF; ++tot)
    {
        len = strlen(str);
        p = r, now = 0;
        for (j = 0;j < len; ++j)
        {
```

```
                w = str[j]-'a';
                if (p->s[w])
                    p = p->s[w], p->w[tot] = max(p->w[tot], ++now);
                else
                {
                    while (p && !p->s[w])
                        p = p->suf;
                    if (!p)
                        p = r, now = 0;
                    else
                        now = p->val+1, p = p->s[w], p->w[tot] = max(p->w[tot], now);
                }
            }
        }
    for (i = L;i >= 0; --i)
        for (node *p = point[i];p;p = p->next)
        {
            res = p->val;
            for (j = 1;j < tot; ++j)
            {
                res = min(p->w[j], res);
                if (p->suf)
                    p->suf->w[j] = max(p->suf->w[j], p->w[j]);
            }
            ans = max(ans, res);
        }
    printf("%d\n", ans);
    return 0;
}
```

## 扩展 KMP

```
//使用getExtend获取extend数组(s[i]...s[n-1]与t的最长公共前缀的长度)
//s,t,slen,tlen,分别为对应字符串及其长度.
//next数组返回t[i]...t[m-1]与t的最长公共前缀长度,调用时需要提前开辟空间
void getNext(char* t, int tlen, int* next)
{
    next[0] = tlen;
    int a;
    int p;
    for (int i = 1, j = -1; i < tlen; i++, j--)
    {
        if (j < 0 || i + next[i - a] >= p)
        {
            if (j < 0) {
                p = i;
                j = 0;
            }
```

```
            while (p < tlen && t[p] == t[j]) {
                p++;
                j++;
            }
            next[i] = j;
            a = i;
        }
        else {
            next[i] = next[i - a];
        }
    }
}

void getExtend(char* s, int slen, char* t, int tlen, int* extend, int* next)
{
    getNext(t, next);
    int a;
    int p;

    for (int i = 0, j = -1; i < slen; i++, j--)
    {
        if (j < 0 || i + next[i - a] >= p)
        {
            if (j < 0) {
                p = i, j = 0;
            }
            while (p < slen && j < tlen && s[p] == t[j]) {
                p++;
                j++;
            }
            extend[i] = j;
            a = i;
        }
        else {
            extend[i] = next[i - a];
        }
    }
}
```

# 杂项

## 测速

```
/*
 require c++11 support
*/
#include <chrono>
using namespace chrono;
```

```cpp
int main(){
    auto start = system_clock::now();
    //do something
    auto end   = system_clock::now();
    auto duration = duration_cast<microseconds>(end - start);
    cout << double(duration.count()) * microseconds::period::num /
        microseconds::period::den << endl;
}
```

## 日期公式

```cpp
/*
    zeller返回星期几%7
*/
int zeller(int y,int m,int d) {
    if (m<=2) y--,m+=12; int c=y/100; y%=100;
    int w=((c>>2)-(c<<1)+y+(y>>2)+(13*(m+1)/5)+d-1)%7;
    if (w<0) w+=7; return(w);
}
/*
    用于计算天数
*/
int getId(int y, int m, int d) {
    if (m < 3) {y --; m += 12;}
    return 365 * y + y / 4 - y / 100 + y / 400 + (153 * m + 2) / 5 + d;
}
```

## 读入挂

```cpp
//  BUF_SIZE对应文件大小
//  调用read(x)或者x=getint()
#define BUF_SIZE 100000
bool IOerror = 0;
inline char nc(){//next char
    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;
    int sgn = 1;
    while(blank(ch = nc()));
    if(IOerror)
        return;
    if(ch=='-')sgn=-1,ch=nc();
    for(x = ch - '0'; (ch = nc()) >= '0' && ch <= '9'; x = x * 10 + ch - '0');
    x*=sgn;
}
inline int getint(){
    int x=0;
    char ch;
    int sgn = 1;
    while(blank(ch = nc()));
    if(IOerror)
        return;
    if(ch=='-')sgn=-1,ch=nc();
    for(x = ch - '0'; (ch = nc()) >= '0' && ch <= '9'; x = x * 10 + ch - '0');
    x*=sgn;
    return x;
}
inline void print(int x){
    if (x == 0){
        puts("0");
        return;
    }
    short i, d[101];
    for (i = 0;x; ++i)
        d[i] = x % 10, x /= 10;
    while (i--)
        putchar(d[i] + '0');
    puts("");
}
#undef BUF_SIZE
```

## 高精度

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>

#include <iostream>
#include <algorithm>

```

```cpp
#include <map>
#include <stack>

typedef long long ll;
typedef unsigned int uint;
typedef unsigned long long ull;
typedef double db;
typedef unsigned char uchar;

using namespace std;
inline bool isnum(char c) { return '0' <= c && c <= '9'; }
inline int getint(int x=0) { scanf("%d", &x); return x; }
inline ll getll(ll x=0) { scanf("%lld", &x); return x; }
double getdb(double x=0) { scanf("%lf",&x); return x; }

//==========================================================

/// 大整数模板.
/// 这个模板保证把一个数字存成 v[0]*SYS^0 + v[1]*SYS^1 +... 的形式.
/// 支持负数运算.
struct bign
{
    static const int SYS = 10; // 多少进制数.
    static const int SIZE = 200; //  数位数.
    int v[SIZE]; // 数位, 从0到N从低到高. 注意可能会爆栈, 可以把它换成指针.
    int len;

    //=====================================================
    //                       工具函数
    //=====================================================

    // 进位和退位整理.
    void Advance(int const& i)
    {
        int k = v[i] / SYS;
        v[i] %= SYS;
        if(v[i] < 0) { k--; v[i] += SYS; }
        v[i+1] += k;
    }

    /// 进位和退位处理. 注意不会减少len.
    void Advance()
    { for(int i=0; i<len; i++) Advance(i); if(v[len] != 0) len++; }

    /// 去除前导0和前导-1.
    void Strip()
    {
        while(len > 0 && v[len-1] == 0) len--;
        while(len > 0 && v[len-1] == -1 && v[len-1] != 0) { len--; v[len] =
            0; v[len-1] -= 10; }
    }
```

```cpp

    bool isNegative() const { return len != 0 && v[len-1] < 0; }

    int& operator[](int const& k) { return v[k]; }

    //==========================================================
    //                          构造函数
    //==========================================================

    // 初始化为0.
    bign() { memset(this, 0, sizeof(bign)); }

    // 从整数初始化.
    bign(ll k)
    {
        memset(this, 0, sizeof(bign));
        while(k != 0) { v[len++] = k % SYS; k /= SYS; }
        Advance();
    }

    // 从字符串初始化. 仅十进制. 支持 -0, 0, 正数, 负数. 不支持前导0, 如 00012, -000, -0101.
    bign(const char* f)
    {
        memset(this,0,sizeof(bign));
        if(f[0] == '-')
        {
            f++;
            int l = strlen(f);
            for(int i=l-1; i>=0; i--) v[len++] = -(f[i] - '0');
            Advance();
            if(len == 1 && v[len-1] == 0) len = 0;
        }
        else
        {
            int l = strlen(f);
            for(int i=l-1; i>=0; i--) v[len++] = f[i] - '0';
            if(len == 1 && v[0] == 0) len--;
        }
    }

    // 拷贝构造函数.
    bign(bign const& f) { memcpy(this, &f, sizeof(bign)); }

    // 拷贝函数.
    bign operator=(bign const& f)
    {
        memcpy(this, &f, sizeof(bign));
        return *this;
    }

```

```cpp
//==========================================================
//                      比较大小
//==========================================================

bool operator==(bign const& f) const
{
    if(len != f.len) return false;
    for(int i=0; i<len; i++) if(v[i] != f.v[i]) return false;
    return true;
}

bool operator<(bign const& f) const
{
    if(isNegative() && !f.isNegative()) return true;
    if(!isNegative() && f.isNegative()) return false;
    if(isNegative() && f.isNegative())
    {
        if(len != f.len) return len > f.len;
        for(int i=len-1; i>=0; i--) if(v[i] != f.v[i]) return v[i] > f.v[i];
        return false;
    }

    if(len != f.len) return len < f.len;
    for(int i=len-1; i>=0; i--) if(v[i] != f.v[i]) return v[i] < f.v[i];
    return false;
}

bool operator>(bign const& f) const { return f < *this; }
bool operator<=(bign const& f) const { return !(*this > f); }
bool operator>=(bign const& f) const { return !(*this < f); }

//==========================================================
//                        运算
//==========================================================

bign operator-() const
{
    bign c = *this;
    for(int i=c.len-1; i>=0; i--) { c[i] = -c[i]; }
    c.Advance();
    c.Strip();
    return c;
}

bign operator+(bign const& f) const
{
    bign c;
    c.len = max(len, f.len);
    for(int i=0; i<c.len; i++) c[i] = v[i] + f.v[i];
    c.Advance();
```

```
        c.Strip();
        return c;
    }

    bign operator-(bign const& f) const { return *this + (-f); }

    bign operator*(int const& k) const
    {
        bign c;
        c.len = len;
        for(int i=0; i<len; i++) c.v[i] = v[i] * k;
        c.len += 10; // 这个乘数需要设置成比 log(SYS, max(k)) 大.
        c.Advance();
        c.Strip();
        return c;
    }

    bign operator*(bign const& f) const
    {
        if(isNegative() && f.isNegative()) return ((-*this) * (-f));
        if(isNegative()) return - ((-*this) * f);
        if(f.isNegative()) return - (*this * (-f));
        bign c;
        c.len = len + f.len;
        for(int i=0; i<len; i++)
        {
            for(int j=0; j<f.len; j++) c[i+j] += v[i] * f.v[j];
            c.Advance();
        }
        c.Strip();
        return c;
    }

    int operator%(int const& k) const
    {
        int res = 0;
        for(int i=len-1; i>=0; i--) (res = res * SYS + v[i]) %= k;
        return res;
    }

    //===========================================================
    //                          输入输出
    //===========================================================

    bign Out(const char* c = "\n") const
    {
        if(len == 0 || (len == 1 && v[0] == 0)) { printf("0%s", c); return *
            this; }
        if(v[len-1] >= 0)
        {
            for(int i=len-1; i>=0; i--) printf("%d", v[i]);
```

```
            printf("%s", c);
        }
        else
        {
            printf("-");
            (-*this).Out(c);
        }
        return *this;
    }

    bign TestOut(const char* c = "\n", int const& sz = 0) const
    {
        printf("[(%d) ", len);
        if(sz == 0) for(int i=0; i<len; i++) printf("%d ", v[i]);
        else for(int i=01; i<sz; i++) printf("%d ", v[i]);
        printf("]\n");
        Out("");
        printf("%s", c);
        return *this;
    }

};
```

## 康托展开与逆展开

```
/// 康托展开.

/// 从一个排列映射到排列的rank.
/// power : 阶乘数组.

/////////////////////////////////////////////////////////////

int power[21];

/// 康托展开, 排名从0开始.
/// 输入为字符串, 其中的字符根据ascii码比较大小.
/// 可以将该字符串替换成其它线序集合中的元素的排列.
int Cantor(const char* c, int len)
{
    int res = 0;
    for(int i=0; i<len; i++)
    {
        int rank = 0;
        for(int j=i; j<len; j++) if(c[j] < c[i]) rank++;
        res += rank * power[len - i - 1];
    }
    return res;
}

```

```
bool cused[21]; // 该数组大小应为字符集的大小.
/// 逆康托展开, 排名从0开始.
/// 输出排名为rank的, 长度为len的排列.
void RevCantor(int rank, char* c, int len)
{
    for(int i=0; i<len; i++) cused[i] = false;
    for(int i=0; i<len; i++)
    {
        int cnt = rank / power[len - i - 1];
        rank %= power[len - i - 1];
        cnt++;
        int num = 0;
        while(true)
        {
            if(!cused[num]) cnt--;
            if(cnt == 0) break;
            num++;
        }
        cused[num] = true;
        c[i] = num + 'a'; // 输出字符串, 从a开始.
    }
}

/// 阶乘数组初始化.
int main()
{
    power[0] = power[1] = 1;
    for(int i=0; i<20; i++) power[i] = i * power[i-1];
    ...
}
```

## 快速乘

```
inline ll mul(ll a,ll b){
    ll d=(ll)floor(a*(double)b/M+0.5);
    ll ret=a*b-d*M;
    if(ret<0)ret+=M;
    return ret;
}
```

## 模拟退火

```
/// 模拟退火.
/// 可能需要魔法调参. 慎用!

/// Tbegin: 退火起始温度.
/// Tend: 退火终止温度.
```

```
/// rate: 退火比率.
/// 退火公式: rand_range(0, 1) > exp(dist / T), 其中 dist 为计算出的优化增量
    .

//////////////////////////////////////////////////////////////////////////

srand(11212);
db Tbegin = 1e2;
db Tend = 1e-6;
db T = Tbegin;
db rate = 0.99995;
int tcnt = 0;
point mvbase = point(0.01, 0.01);
point curp = p[1];
db curmax = GetIntArea(curp);
while(T >= Tend)
{
    // 生成一个新的解.
    point nxtp = curp + point(
        (randdb() - 0.5) * 2.0 * mvbase.x * T,
        (randdb() - 0.5) * 2.0 * mvbase.y * T);

    // 计算这个解的价值.
    db v = GetIntArea(nxtp);

    // 算出距离当前最优解有多远.
    db dist = v - curmax;
    if(dist > eps || (dist < -eps && randdb() > exp(dist / T)))
    {
        // 更新方案和答案.
        curmax = v;
        curp = nxtp;
        tcnt++;
    }

    T *= rate;
}
```

## 常用概念

### 映射

[injective] or [one-to-one] 函数值不重复

[surjective] or [onto] 值域都被取到

[bijective] or [one-to-one correspondence] 一一对应

## 反演

反演中心 $O$, 反演半径 $r$, 点 $p$ 的反演点 $p'$ 满足 $|OP||OP'| = r^2$

不经过反演中心的直线，反形为经过反演中心的圆

不经过反演中心的圆，反形为圆，反演中心为这两个互为反形的圆的位似中心

## 弦图

设 $next(v)$ 表示 $N(v)$ 中最前的点. 令 $w*$ 表示所有满足 $A \in B$ 的 $w$ 中最后的一个点, 判断 $v \cup N(v)$ 是否为极大团, 只需判断是否存在一个 $w \in w*$, 满足 $Next(w) = v$ 且 $|N(v)| + 1 \leq |N(w)|$ 即可.

## 五边形数

$\prod_{n=1}^{\infty}(1 - x^n) = \sum_{n=0}^{\infty}(-1)^n(1 - x^{2n+1})x^{n(3n+1)/2}$

## 重心

半径为 $r$，圆心角为 $\theta$ 的扇形重心与圆心的距离为 $\frac{4r\sin(\theta/2)}{3\theta}$

半径为 $r$，圆心角为 $\theta$ 的圆弧重心与圆心的距离为 $\frac{4r\sin^3(\theta/2)}{3(\theta-\sin(\theta))}$

## 第二类 Bernoulli number

$B_m = 1 - \sum_{k=0}^{m-1}\binom{m}{k}\frac{B_k}{m-k+1}$

$S_m(n) = \sum_{k=1}^{n}k^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_k n^{m+1-k}$

## Fibonacci 数

$F_n = \frac{\varphi^n - (-\varphi)^{-n}}{\sqrt{5}}, \varphi = \frac{1+\sqrt{5}}{2}$

$F_n = \lfloor \frac{\varphi^n}{\sqrt{5}} + \frac{1}{2} \rfloor$

## Catalan 数

$C_{n+1} = \frac{2(2n+1)}{n+2}C_n$

$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$

前 20 项:1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190

所有的奇卡塔兰数 $C_n$ 都满足 $n = 2^k - 1$。所有其他的卡塔兰数都是偶数

## Stirling 数

第一类:n 个元素的项目分作 k 个环排列的方法数目

$s(n,k) = (-1)^{n+k}|s(n,k)|$

$|s(n,0)| = 0$

$|s(1,1)| = 1$

$|s(n,k)| = |s(n-1,k-1)| + (n-1) * |s(n-1,k)|$

第二类:n 个元素的集定义 k 个等价类的方法数

$S(n,1) = S(n,n) = 1$

$S(n,k) = S(n-1,k-1) + k * S(n-1,k)$

## 三角公式

$\sin(a \pm b) = \sin a \cos b \pm \cos a \sin b$

$\cos(a \pm b) = \cos a \cos b \mp \sin a \sin b$

$\tan(a \pm b) = \frac{\tan(a) \pm \tan(b)}{1 \mp \tan(a)\tan(b)}$

$\tan(a) \pm \tan(b) = \frac{\sin(a \pm b)}{\cos(a)\cos(b)}$

$\sin(a) + \sin(b) = 2\sin(\frac{a+b}{2})\cos(\frac{a-b}{2})$

$\sin(a) - \sin(b) = 2\cos(\frac{a+b}{2})\sin(\frac{a-b}{2})$

$\cos(a) + \cos(b) = 2\cos(\frac{a+b}{2})\cos(\frac{a-b}{2})$

$\cos(a) - \cos(b) = -2\sin(\frac{a+b}{2})\sin(\frac{a-b}{2})$

$\sin(na) = n\cos^{n-1} a \sin a - \binom{n}{3}\cos^{n-3} a \sin^3 a + \binom{n}{5}\cos^{n-5} a \sin^5 a - \dots$

$\cos(na) = \cos^n a - \binom{n}{2}\cos^{n-2} a \sin^2 a + \binom{n}{4}\cos^{n-4} a \sin^4 a - \dots$