Standard Code Library
of
Eating Keyboard

Edited By

sddyzjh
DragoonKiller
Alisa

Huazhong University of Science and Technology

2018.09.14

目录

计算几何

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); }
    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;
```

```cpp
    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

```cpp
/// 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

```cpp
/// 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 *.
```

可持久化并查集

```cpp
int n,m,sz;
int root[200005],ls[2000005],rs[2000005],v[2000005],deep[2000005];
void build(int &k,int l,int r){
    if(!k)k=++sz;
    if(l==r){v[k]=l;return;}
    int mid=(l+r)>>1;
    build(ls[k],l,mid);
    build(rs[k],mid+1,r);
}
void modify(int l,int r,int x,int &y,int pos,int val){
    y=++sz;
    if(l==r){v[y]=val;return;}
    ls[y]=ls[x];rs[y]=rs[x];
    int mid=(l+r)>>1;
    if(pos<=mid)
        modify(l,mid,ls[x],ls[y],pos,val);
    else modify(mid+1,r,rs[x],rs[y],pos,val);
}
int query(int k,int l,int r,int pos){
    if(l==r)return k;
    int mid=(l+r)>>1;
    if(pos<=mid)return query(ls[k],l,mid,pos);
    else return query(rs[k],mid+1,r,pos);
}
void add(int k,int l,int r,int pos){
    if(l==r){deep[k]++;return;}
    int mid=(l+r)>>1;
    if(pos<=mid)add(ls[k],l,mid,pos);
    else add(rs[k],mid+1,r,pos);
```

```
}
int find(int k,int x){
    int p=query(k,1,n,x);
    if(x==v[p])return p;
    return find(k,v[p]);
}
int la=0;
int main(){
    n=read();m=read();
    build(root[0],1,n);
    int f,k,a,b;
    for(int i=1;i<=m;i++){
        f=read();
        if(f==1){//合并
            root[i]=root[i-1];
            a=read()^la;b=read()^la;
            int p=find(root[i],a),q=find(root[i],b);
            if(v[p]==v[q])continue;
            if(deep[p]>deep[q])swap(p,q);
            modify(1,n,root[i-1],root[i],v[p],v[q]);
            if(deep[p]==deep[q])add(root[i],1,n,v[q]);
        }
        if(f==2)//返回第 k 次的状态
        {k=read()^la;root[i]=root[k];}
        if(f==3){//询问
            root[i]=root[i-1];
            a=read()^la;b=read()^la;
            int p=find(root[i],a),q=find(root[i],b);
            if(v[p]==v[q])puts("1"),la=1;
            else puts("0"),la=0;
        }
    }
    return 0;
}
```

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;

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


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
#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
*/
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[22][2][10];
int digit[20];
//pos: 当前位置;lim: 是否考虑位数;pre: 前一位;alr: 已经匹配?
int dps(int pos, int lim, int pre, int alr){
    if(pos < 0){
        return alr;
    }
    if(!lim && (dp[pos][alr][pre] != -1)){
        return dp[pos][alr][pre];
    }
```

```
    int result = 0;
    int len = lim ? digit[pos] : 9;
    for(int i = 0; i <= len; i++){
        result += dps(pos - 1, lim && (i == len), i, alr || (pre == 6 && i == 2)||(i==4));
    }
    if(!lim){
        dp[pos][alr][pre] = result;
    }
    return result;
}
int solve(int x){
    memset(dp, -1, sizeof(dp));
    int length = 0;
    while(x){
        digit[length++] = (x % 10);
        x /= 10;
    }
    return dps(length - 1, 1, 0, 0);
}
int main(){
    int a,b;
    while(scanf("%d%d",&a,&b),a||b){
        printf("%d\n", b-a+1-slove(b>0?b:1)+slove((a-1)>0?(a-1):1));
    }
    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
*/
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
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
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

```
//f[i] 表示前缀 LIS,g[i] 表示长为 i 的 LIS 的最小结尾数字
int LIS(int *f, int *g){
	memset(f,0,(n+1)*sizeof(int));
	f[1] = 1;
	memset(g,127,(n+1)*sizeof(int));
	g[0] = -infi;
	int nmax = 1;
	g[nmax] = a[1];
	rep(i,2,n){
		int v = lower_bound(g,g+nmax+1,a[i])-g-1;
		f[i] = v+1;
		nmax = max(nmax, v+1);
		g[v+1] = min(g[v+1], a[i]);
	}
	return nmax;
}
```

图论

k 短路可持久化堆

```
/*
       G 为原图, E 为反图
       细节看 solve()
*/
namespace Leftist_Tree{
    struct Node{
        int l, r, x, h;
        int val;
    }T[N*50];
    int Root[N];
    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 = infi;
    }
    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, int val, int v){
        int o = newnode(T[0]);
        T[o].val = val, T[o].x = v;
        x = merge(x, o);
    }
}
using namespace Leftist_Tree;
struct Edge{
        int v, w, n;
}G[N], E[N];
int cnt, point[N], cnt1, point1[N];
void adde(int u, int v, int w = 0){
        G[++cnt]=(Edge){v,w,point[u]},point[u]=cnt;
        E[++cnt1]=(Edge){u,w,point1[v]},point1[v]=cnt1;
}
int n, m, Len;
void Ginit(){
        cnt = cnt1 = 0;
        fill(point,0,n+1);
        fill(point1,0,n+1);
}
int vis[N];
int in[N], p[N];
int d[N];
void dij(int s){
    priority_queue<pii> q;
    d[s] = 0;
    q.push(mp(0, s));
    while(!q.empty()){
        int u = q.top().se;
        q.pop();
        if(vis[u])continue;
        vis[u] = 1;
        for(int i = point1[u];i;i=E[i].n){
            int v = E[i].v;
            if(d[v]> d[u] + E[i].w){
                p[v] = u;
                d[v] = d[u] + E[i].w;
                q.push(mp(-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 = point[u];i;i=G[i].n){
        int v = G[i].v;
        if(d[v] == infi)continue;
        if(p[u] == v && d[u] == G[i].w + d[v] && flag){
            flag = 0;
            continue;
        }
        int val = d[v] - d[u] + G[i].w;
        insert(Root[u], val, v);
    }
    for(int i = point1[u];i;i=E[i].n){
        if(p[E[i].v] == u)dfs(E[i].v);
    }
}

int kth(int s, int t, int k){
    dij(t);
    if(d[s] == infi){
        return -1;
    }
    if(s != t)--k;
    if(!k){
                return -1;
    }
    fill(vis,0,n+1);
    init();
    Root[t] = 0;
    dfs(t);
    priority_queue<pii, vector<pii>, greater<pii>> q;
    if(Root[s])q.push(mp(d[s] + T[Root[s]].val, Root[s]));
    while(k--){
        if(q.empty()){
            return -1;
        }
        pii u = q.top();
        q.pop();
        if(!k){
                        return u.fi;
        }
        int x = T[u.se].l, y = T[u.se].r, v = T[u.se].x;
        if(Root[v])q.push(mp(u.fi + T[Root[v]].val, Root[v]));
        if(x)q.push(mp(u.fi + T[x].val - T[u.se].val, x));
        if(y)q.push(mp(u.fi + T[y].val - T[u.se].val, y));
    }
}

void solve(){
        Ginit();
    rep(i,1,n){
        d[i] = infi;
        vis[i] = 0;
        p[i] = 0;
    }
    int s, t, k;
    sc(s),sc(t),sc(k),sc(Len);
    rep(i,1,m){
        int u, v, c;
        sc(u),sc(v),sc(c);
        adde(u, v, c);
    }
    int res = kth(s,t,k);
    if(res >= 0 && res <= Len)
                printf("yareyaredawa\n");
        else
                printf("Whitesnake!\n");
}

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);
	rep(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, num[N];
struct E{
        int u, v, n;
}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].n){
                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]++;
                }
                while (v != u);
        }
}
void Ginit(){
        cnt = 0;
        fill(point,0,n+1);
}
void SCC(){
        Stop = sccnum = dfsnow = 0;
        fill(dfn, 0, n+1);
        rep(i,1,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 degre[N];
void suodian(){
        Ginit();
        fill(degre,0 ,n+1);
        rep(i,1,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);
        rep(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 lca_init() 后
	调用 lca(u,v) 得到 u,v 的 lca
*/
int fa[N][M];
void lca_init(){
	rep(k,1,M-1)rep(i,1,n)
		fa[i][k] = fa[fa[i][k-1]][k-1];
}
int lca(int u,int v){
	if(dep[u] < dep[v])
		swap(u, v);
	repr(i,0,M-1)
		if(((dep[u] - dep[v])>>i) & 1)
			u = fa[u][i];
	repr(i,0,M-1)
		if(fa[u][i] != fa[v][i]){
			u = fa[u][i];
			v = fa[v][i];
		}
	if(u != v)
		return fa[u][0];
	return u;
}
```

点分治

```
int n, siz[N], maxs[N], r;
bitset<N> vis;
void getroot(int u, int f){
    siz[u] = 1, maxs[u] = 0;
    for (int i = point[u];i;i = G[i].n){
        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;
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].n){
            if (hh[G[p].v] || vis[G[p].v])continue;
            Q.push(G[p].v);
        }
    }
}
int calc(int u){
	int res(0);
    bfs(u);
    return res;
}
void solve(int u){
    dis[u] = 0, vis[u] = 1;
    ans += calc(u);
    for (int i = point[u];i;i = G[i].n){
        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);
    }
}
void p_d(){
	vis.reset();
```

```
        maxs[r=0]=n+1;
    getroot(1, 0);
    solve(r);
}
```

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;
    }
};
bool vis[N];
int dist[N];
void dij(int s){
    fill(vis,0,n+1);
    fill(dist,127,n+1);
        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]});
            }
        }
    }
}
```

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,1,n)
                rep(j,1,n)
                        a[i][j]=(a[i][j]+mod)%mod;
        ll ans=1,f=1;
        rep(i,1,n){
                rep(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);
                                rep(k,i,n)
                                        a[i][k]=(a[i][k]-t*a[j][k]%mod+mod)%mod;
                                rep(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

```
#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;
        fill(point,0,T+1);
}
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;
}
int BFS(){
        queue<int> q;
        fill(D,0,T+1);
        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;
                                        if(v==T)return D[T];
                                        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;
                        }
                }
        }
        if(!f)D[u]=0;
        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;
        }
```

```cpp
	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

```cpp
/*
	只限制一个点的度数
```

```
*/
#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

```cpp
#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;
}
```

欧拉路径覆盖

```cpp
/// 无向图的最少欧拉路径覆盖
/// mxn : 点数.
/// mxm : 边数.
/// 最终结果存在 ls 中, 代表边的编号.
/// 初始化: 直接将图和结果链表清除即可.

// et = pool;
// memset(eds, 0, sizeof(edge*) * (n + 1));
// ls.clear();

/// AC HDU 6311

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

typedef list<edge*>::iterator iter;

const int mxn = 1e5 + 50;
cosnt int mxm = 1e5 + 50;
struct edge { int id; int in; edge* nxt; bool used; } pool[mxm * 2]; edge* et = pool;
```

```
edge* opp(edge* t) { int x = (int)(t - pool); if(x & 1) return t - 1; return t + 1; }
edge* eds[mxn]; // 注意这一数组在运算时可能改变. 需要原图的话应做备份.
void addedge(int a, int b, int id)
{
    et->used = false; et->id = id; et->in = b; et->nxt = eds[a]; eds[a] = et++;
    et->used = false; et->id = -id; et->in = a; et->nxt = eds[b]; eds[b] = et++;
}
int n, m;
int deg[mxn]; //度数.
list<edge*> ls;
iter pos[mxn];
bool inq[mxn];
queue<int> q;
int stk[mxn]; int st = 0;
// 走一条路, 清除路上的边.
// 如果起点是奇数度, 最终会走到另一个度数为奇数的点.
// 如果起点是偶数度, 最终会走回起点.
void Reduce(int x, iter loc)
{
    stk[st++] = x;
    while(true)
    {
        while(eds[x] && eds[x]->used) eds[x] = eds[x]->nxt;
        if(!eds[x]) break;
        edge* e = eds[x];
        opp(e)->used = true;
        e->used = true;
        deg[x]--;
        deg[e->in]--;
        pos[x] = ls.insert(loc, e);
        x = stk[st++] = e->in;
    }
    repr(i, 0, st-1) if(deg[stk[i]] != 0 && !inq[stk[i]])
    {
        q.push(stk[i]);
        inq[stk[i]] = true;
    }
    st = 0;
}
// 使用欧拉路清除同一个连通分量内部的边.
void ReduceIteration()
{
    while(!q.empty())
    {
        int x = q.front(); q.pop(); inq[x] = false;
        if(deg[x] & 1)
        {
            Reduce(x, ls.end());
            ls.insert(ls.end(), nullptr);
        }
        else if(deg[x] != 0)  Reduce(x, pos[x]);
    }
}

......

{
    // 读入数据.
    rep(i, 1, m)
    {
        int a = getint();
        int b = getint();
        deg[a]++;
        deg[b]++;
        addedge(a, b, i);
    }

    // 初始化.
    rep(i, 1, n) pos[i] = ls.end();

    // 先清除所有奇数度节点所在联通块.
    rep(i, 1, n) if(deg[i] & 1) q.push(i);
    ReduceIteration();

    // 清除所有仅包含偶数度节点的联通块.
    rep(i, 1, n) if(deg[i] != 0)
    {
```

```
            q.push(i);
            inq[i] = true;
            ReduceIteration();
            ls.insert(ls.end(), nullptr);
        }
}
```

数学

常见积性函数

单位函数 $e(x) = \begin{cases} 1, x = 1 \\ 0, x > 1 \end{cases}$

常函数 $I(x) = 1$

幂函数 $id(x) = x^k$

欧拉函数 $\varphi(x) = x\prod_{p|x}(1 - \frac{1}{p})$

$n \geq 2$ 时 $\varphi(n)$ 为偶数

莫比乌斯函数 $\mu(x) = \begin{cases} 1, x = 1 \\ (-1)^k, x = p_1p_2...p_k \\ 0, others \end{cases}$

$\sigma_k(n) = \Sigma_{d|n}d^k$

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

常用公式

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

$[n = 1] = \Sigma_{d|n}\mu(d)$ 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

狄利克雷卷积

$h(n) = \sum_{d|n} f(d)g(\frac{n}{d})$ 称为 f 和 g 的狄利克雷卷积，也可以理解为 $h(n) = \sum_{ij=n} f(i)g(j)$

两个积性函数的狄利克雷卷积仍为积性函数

狄利克雷卷积满足交换律和结合律

莫比乌斯反演

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

即 $f = g * I \Leftrightarrow g = \mu * f$

$\mu * I = e$

$f = g * I \Rightarrow \mu * f = g * (\mu * I) = g * e = g$

$g = \mu * f \Rightarrow f = g * I$

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

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

常用等式

$\varphi = \mu * id$

$\varphi * I = id$

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

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

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

常用代换

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

考虑每个数的贡献

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

Pell 方程

形如 $x^2 - dy^2 = 1$ 的方程

当 d 为完全平方数时无解

假设 (x_0, y_0) 为最小正整数解

$x_n = x_{n-1} \times x_0 + d \times y_{n-1} \times y_0$

$y_n = x_{n-1} \times y_0 + y_{n-1} \times x_0$

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 c[MATN][MATN];
	mat(){SET(c,0);}
};
mat cheng(const mat &a, const mat &b){
	mat w = mat();
	rep(i,0,MATN-1)rep(j,0,MATN-1)rep(k,0,MATN-1){
		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 = mat();
	rep(j,0,MATN-1)
		i.c[j][j] = 1;
	if(k<0)return i;
	while(k){
		if(k&1)
			i=cheng(i,a);
		a=cheng(a,a);
		k>>=1;
	}
	return i;
}
```

线性规划

```
//求 max{cx|Ax<=b,x>=0} 的解
typedef vector<double> VD;
VD simplex(vector<VD> A, VD b, VD c) {
	int n = A.size(), m = A[0].size() + 1, r = n, s = m - 1;
	vector<VD> D(n + 2, VD(m + 1, 0)); vector<int> ix(n + m);
	for (int i = 0; i < n + m; ++ i) ix[i] = i;
	for (int i = 0; i < n; ++ i) {
		for (int j = 0; j < m - 1; ++ j) D[i][j] = -A[i][j];
		D[i][m - 1] = 1; D[i][m] = b[i];
		if (D[r][m] > D[i][m]) r = i;
	}
	for (int j = 0; j < m - 1; ++ j) D[n][j] = c[j];
	D[n + 1][m - 1] = -1;
	for (double d; ; ) {
		if (r < n) {
			int t = ix[s]; ix[s] = ix[r + m]; ix[r + m] = t;
			D[r][s] = 1.0 / D[r][s]; vector<int> speedUp;
			for (int j = 0; j <= m; ++ j) if (j != s) {
				D[r][j] *= -D[r][s];
				if(D[r][j]) speedUp.push_back(j);
			}
			for (int i = 0; i <= n + 1; ++ i) if (i != r) {
				for(int j = 0; j < speedUp.size(); ++ j)
				D[i][speedUp[j]] += D[r][speedUp[j]] * D[i][s];
				D[i][s] *= D[r][s];
		}} r = -1; s = -1;
		for (int j = 0; j < m; ++ j) if (s < 0 || ix[s] > ix[j])
			if (D[n + 1][j] > EPS || (D[n + 1][j] > -EPS && D[n][j] > EPS)) s = j;
		if (s < 0) break;
		for (int i = 0; i < n; ++ i) if (D[i][s] < -EPS)
			if (r < 0 || (d = D[r][m] / D[r][s] - D[i][m] / D[i][s]) < -EPS
					|| (d < EPS && ix[r + m] > ix[i + m])) r = i;
		if (r < 0) return VD(); // 无边界
	}
	if (D[n + 1][m] < -EPS) return VD(); // 无解
	VD x(m - 1);
	for (int i = m; i < n + m; ++ i) if (ix[i] < m - 1) x[ix[i]] = D[i - m][m];
	return x; // 最优值在 D[n][m]
}
```

线性基

```
ll a[N],b[N];
// 插入一个数
void insert(ll *ff,ll x){
    repr(i,0,60)
        if((x>>i)&1ll)
            if(!ff[i]){
                ff[i] = x;
                return;
            }
            else
                x ^= ff[i];
}
// 查询一个数是否在异或集合内
bool check(ll x){
    repr(i,0,60){
        if((x>>i)&1){
            if(((b[i]>>i)&1)==0)return 0;
            x^=b[i];
            if(!x)return 1;
        }
    }
    return 0;
}
```

线性筛

```
/*
        is=0 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;
bool is[N];
int phi[N];
int mu[N];
int minp[N];
int mina[N];
int d[N];
void getprime(int list){
        mu[1] = 1;
        phi[1] = 1;
        is[1] = 1;
        rep(i,2,list){
                if(!is[i]){
                        // 新的质数
                        prime[++size] = i;
                        phi[i] = i-1;
                        mu[i] = -1;
                        minp[i] = i;
                        mina[i] = 1;
                        d[i] = 2;
                }
                rep(j,1,size){
                        // 用已有的质数去筛合数
                        if(i*prime[j]>list)
                                break;
                        // 标记合数
                        is[i * prime[j]] = 1;
                        minp[i*prime[j]] = prime[j];
                        if(i % prime[j] == 0){
                                // i 是质数的倍数
                                // 这个质数的次数大于 1
                                mu[i*prime[j]] = 0;
                                // 次数 ++
                                phi[i*prime[j]] = phi[i] * prime[j];
                                // 次数 ++
                                mina[i*prime[j]] = mina[i]+1;
                                d[i*prime[j]] = d[i]/(mina[i]+1)*(mina[i]+2);
                                break;
```

```
            }else{
                // 添加一个新的质因子
                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]];
            }
        }
    }
}
```

线性求逆元

```
inv[1] = 1;
rep(i,2,n)inv[i] = (MOD-(MOD/i)) * (ll)inv[MOD%i] % MOD;
```

FFT

```
#define maxfft 524288+5
const double pi=acos(-1.0);
struct cp{
    double a,b;
    cp operator +(const cp &o)const {return (cp){a+o.a,b+o.b};}
    cp operator -(const cp &o)const {return (cp){a-o.a,b-o.b};}
    cp operator *(const cp &o)const {return (cp){a*o.a-b*o.b,b*o.a+a*o.b};}
    cp operator *(const double &o)const {return (cp){a*o,b*o};}
    cp operator !() const{return (cp){a,-b};}
}w[maxfft];
int pos[maxfft];
void fft_init(int len){
    int j=0;
    while((1<<j)<len)j++;
    j--;
    for(int i=0;i<len;i++)
        pos[i]=pos[i>>1]>>1|((i&1)<<j);
}
void fft(cp *x,int len,int sta){
    for(int i=0;i<len;i++)
        if(i<pos[i])swap(x[i],x[pos[i]]);
    w[0]=(cp){1,0};
    for(unsigned i=2;i<=len;i<<=1){
        cp g=(cp){cos(2*pi/i),sin(2*pi/i)*sta};
        for(int j=i>>1;j>=0;j-=2)w[j]=w[j>>1];
        for(int j=1;j<i>>1;j+=2)w[j]=w[j-1]*g;
        for(int j=0;j<len;j+=i){
            cp *a=x+j,*b=a+(i>>1);
            for(int l=0;l<i>>1;l++){
                cp o=b[l]*w[l];
                b[l]=a[l]-o;
                a[l]=a[l]+o;
            }
        }
    }
    if(sta==-1)for(int i=0;i<len;i++)x[i].a/=len,x[i].b/=len;
}
cp x[maxfft],y[maxfft],z[maxfft];
// a[0..n-1] 和 b[0..m-1] 的卷积存在 c 中
void FFT(int *a,int n,int *b,int m,ll *c){
    int len=1;
    while(len<(n+m+1)>>1)len<<=1;
    fft_init(len);
    for(int i=n/2;i<len;i++)x[i].a=x[i].b=0;
    for(int i=m/2;i<len;i++)y[i].a=y[i].b=0;
    for(int i=0;i<n;i++)(i&1?x[i>>1].b:x[i>>1].a)=a[i];
    for(int i=0;i<m;i++)(i&1?y[i>>1].b:y[i>>1].a)=b[i];
    fft(x,len,1),fft(y,len,1);
    for(int i=0;i<len/2;i++){
        int j=len-1&len-i;
        z[i]=x[i]*y[i]-(x[i]-!x[j])*(y[i]-!y[j])*(w[i]+(cp){1,0})*0.25;
    }
    for(int i=len/2;i<len;i++){
        int j=len-1&len-i;
        z[i]=x[i]*y[i]-(x[i]-!x[j])*(y[i]-!y[j])*((cp){1,0}-w[i^len>>1])*0.25;
    }
```

```
    fft(z,len,-1);
    for(int i=0;i<n+m;i++)
        if(i&1)c[i]=(ll)(z[i>>1].b+0.5);
        else c[i]=(ll)(z[i>>1].a+0.5);
}
```

NTT+CRT

```
/*
        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 convolution(a,n,b,m,c)
        MOD 为模数,CRT 合并
        若模数为 m1, 卷积做到 x3, 把 x3 替换为 c
        首先调用 GetWn(m1,WN[0]),GetWn(m2,WN[1])
        模数满足的性质为 mod=2^k*(奇数)+1 2^k>2n 时可以在模意义下做 FFT
        998244353 = 2^23*7*17+1
        1004535809 = 2^21*479+1
*/
const int G = 3;
const int MOD=1000003,m1=998244353,m2=1004535809;
const ll P=1002772198720536577LL;
inline ll mul(ll a,ll b){
        ll d=(ll)floor(a*(double)b/P+0.5);
        ll ret=a*b-d*P;
        if(ret<0)ret+=P;
        return ret;
}
inline int CRT(int r1,int r2){
        ll a = mul(r1,m2);
        a = mul(a,332747959ll);
        ll b = mul(r2,m1);
        b = mul(b,669690699ll);
        a = (a+b)%P;
        return a%MOD;
}
int mul(int x, int y, int mod){
        ll z = 1LL*x*y;
        return z-z/mod*mod;
}
int add(int x, int y, int mod){
        x += y;
        if(x >= mod)x -= mod;
        return x;
}
const int NUM = 20;
int WN[2][NUM];
void GetWn(int mod, int wn[]){
        rep(i,0,NUM-1){
                int t = 1<<i;
                wn[i] = pwM(G, (mod - 1) / t, mod);
        }
}
void NTT(int a[], int len, int t, int mod, int wn[]){
        for(int i = 0,j = 0;i < len; ++i){
                if(i > j)swap(a[i], a[j]);
                for(int l = len >> 1;(j ^= l) < l;l >>= 1);
        }
        int id = 0;
        for(int h = 2;h <= len;h <<= 1){
                id++;
                for(int j = 0;j < len;j += h){
                        int w = 1;
                        for(int k = j;k < j+h/2; ++k){
                                int u = a[k];
                                int t = mul(w, a[k+h/2], mod);
                                a[k] = add(u, t, mod);
                                a[k+h/2] = add(u, mod-t, mod);
                                w = mul(w, wn[id], mod);
                        }
                }
        }
        if(t == -1){
                rep(i,1,len/2-1)swap(a[i], a[len-i]);
                int inv = pwM(len, mod-2, mod);
```

```
                rep(i,0,len-1)a[i] = mul(a[i], inv, mod);
        }
}
int x1[N], x2[N], x3[N], x4[N];
void convolution(ll a[], int l1, ll b[], int l2, ll c[]){
        int len = 1;
        while(len < l1*2 || len < l2*2)len <<= 1;
        rep(i,0,l1-1)x1[i] = a[i]%m1;
        rep(i,l1,len-1)x1[i] = 0;
        rep(i,0,l2-1)x2[i] = b[i]%m1;
        rep(i,l2,len-1)x2[i] = 0;
        NTT(x1,len,1,m1,WN[0]);NTT(x2,len,1,m1,WN[0]);
        rep(i,0,len-1)x3[i] = (ll)x1[i]*x2[i]%m1;
        NTT(x3,len,-1,m1,WN[0]);
        // 单模数到这里结束
        rep(i,0,l1-1)x1[i] = a[i]%m2;
        rep(i,l1,len-1)x1[i] = 0;
        rep(i,0,l2-1)x2[i] = b[i]%m2;
        rep(i,l2,len-1)x2[i] = 0;
        NTT(x1,len,1,m2,WN[1]);NTT(x2,len,1,m2,WN[1]);
        rep(i,0,len-1)x4[i] = (ll)x1[i]*x2[i]%m2;
        NTT(x4,len,-1,m2,WN[1]);
        // 合并两次卷积的结果
        rep(i,0,len-1)c[i] = CRT(x3[i], x4[i]);
}
```

FWT

```
void fwt1(int *a, int len){
        for(int i=0;i<len;i+=2)
                _add(a[i+1],a[i]);
        for(int i=4;i<=len;i<<=1)
                for(int j=0;j<len;j+=i)
                        for(int k=0;k<i/2;k+=2){
                                _add(a[j+k+i/2],a[j+k]);
                                _add(a[j+k+i/2+1],a[j+k+1]);
                        }
}
void fwt2(int *a, int len){
        for(int i=0;i<len;i+=2)
                _sub(a[i+1],a[i]);
        for(int i=4;i<=len;i<<=1)
                for(int j=0;j<len;j+=i)
                        for(int k=0;k<i/2;k+=2){
                                _sub(a[j+k+i/2],a[j+k]);
                                _sub(a[j+k+i/2+1],a[j+k+1]);
                        }
}
void fwt3(int *a, int len){
        for(int i=2;i<=len;i<<=1)
                for(int j=0;j<len;j+=i)
                        for(int k=0;k<i/2;k++){
                                int u=a[j+k];
                                int v=a[j+k+i/2];
                                _add(a[j+k],v);
                                _sub(u,v);
                                a[j+k+i/2]=u;
                        }
}
void fwt4(int *a, int len){
        for(int i=2;i<=len;i<<=1)
                for(int j=0;j<len;j+=i)
                        for(int k=0;k<i/2;k++){
                                int u=a[j+k];
                                int v=a[j+k+i/2];
                                _add(a[j+k],v);
                                _sub(u,v);
                                a[j+k+i/2]=u;
                        }
        ll inv=pw(len%MOD,MOD-2);
        for(int i=0;i<len;i++)
                _mul(a[i],inv);
}
void fwt5(int *a, int len){
        for(int i=2;i<=len;i<<=1)
```

```
			for(int j=0;j<len;j+=i)
				for(int k=0;k<i/2;k++)
					_add(a[j+k],a[j+k+i/2]);
}
void fwt6(int *a, int len){
	for(int i=2;i<=len;i<<=1)
		for(int j=0;j<len;j+=i)
			for(int k=0;k<i/2;k++)
				_sub(a[j+k],a[j+k+i/2]);
}
int bitcount[N];
int a1[18][N],a2[18][N];
void or_conv(int *a,int *b,int *c, int len){
	for(int i=0;i<len;i++)
		a1[bitcount[i]][i]=a[i];
	int width=bitcount[len-1];
	for(int i=0;i<=width;i++)
		fwt1(a1[i],len);
	for(int i=width;i>=0;i--)
		for(int j=0;j<=i;j++)
			for(int k=0;k<len;k++)
				a2[i][k]=(a2[i][k]+(ll)a1[i-j][k]*a1[j][k])%MOD;
	for(int i=0;i<=width;i++)
		fwt2(a2[i],len);
	for(int i=0;i<len;i++)
		c[i]=a2[bitcount[i]][i];
}
void xor_conv(int *a,int *b,int *c, int len){
	static int a1[N],a2[N];
	memcpy(a1,a,sizeof a1);
	memcpy(a2,b,sizeof a2);
	fwt3(a1,len);
	fwt3(a2,len);
	for(int i=0;i<len;i++)
		a1[i]=(ll)a1[i]*a2[i]%MOD;
	fwt4(a1,len);
	memcpy(c,a1,sizeof a1);
}
void and_conv(int *a,int *b,int *c, int len){
	static int a1[N],a2[N];
	memcpy(a1,a,sizeof a1);
	memcpy(a2,b,sizeof a2);
	fwt5(a1,len);
	fwt5(a2,len);
	for(int i=0;i<len;i++)
		a1[i]=(ll)a1[i]*a2[i]%MOD;
	fwt6(a1,len);
	memcpy(c,a1,sizeof a1);
}
```

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];
    }
};
```

子串 Hash

```
/// 字符串/数字串双模哈希.
/// 另外一些大质数, 可以用来做更多模数的哈希.
/// 992837513, 996637573, 996687641, 996687697, 996687721
////////////////////////////////////////////////////////////////////////////////
const int mxn = 1e6 + 50;
const int hashmod1 = 1000000007;
const int hashmod2 = 992837507;
const int sysnum1 = 31;
const int sysnum2 = 29;
ll hx[mxn];
ll hy[mxn];
struct Hash { int x; int y; };
bool operator<(Hash const& a, Hash const& b) { return a.x == b.x ? a.y < b.y : a.x < b.x; }
bool operator==(Hash const& a, Hash const& b) { return a.x == b.x && a.y == b.y; }
bool operator!=(Hash const& a, Hash const& b) { return !(a == b); }
/// 取子串的哈希值. 自觉改值域, 进制数和串类型.
Hash GetHash(int* c, int l, int r)
{
    Hash v = {0, 0};
    rep(i, l, r)
    {
        v.x = (int)(((((ll)v.x * sysnum1) % hashmod1) + c[i] + 1) % hashmod1);
        v.y = (int)(((((ll)v.y * sysnum2) % hashmod2) + c[i] + 1) % hashmod2);
    }
    return v;
}
/// 合并两个串的哈希值. 注意左右顺序.
Hash MergeHash(Hash left, Hash right, int rightLen)
{
    return Hash {
```

```
        (int)(((ll)left.x * hx[rightLen] % hashmod1  + right.x) % hashmod1),
        (int)(((ll)left.y * hy[rightLen] % hashmod2  + right.y) % hashmod2),
    };
}
/// 哈希计算初始化.
void HashInit(int sz)
{
    hx[0] = hy[0] = 1;
    rep(i, 1, sz)
    {
        hx[i] = hx[i-1] * sysnum1 % hashmod1;
        hy[i] = hy[i-1] * sysnum2 % hashmod2;
    }
}
```

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

```
/*
        细节看 main
        最后结果存在 p 为下标的数组内
*/
int n;
int a[N],v[N],h[N],sa[2][N],rk[2][N];
int p,q,k;
void init(){
        SET(a,0);
        SET(v,0);
        SET(h,0);
        SET(sa,0);
        SET(rk,0);
}
void calsa(int *sa,int *rk,int *SA,int *RK){
        rep(i,1,n)v[rk[sa[i]]]=i;
        repr(i,1,n)
                if(sa[i]>k)
                        SA[v[rk[sa[i]-k]]--]=sa[i]-k;
        rep(i,n-k+1,n)SA[v[rk[i]]--]=i;
        rep(i,1,n)
                RK[SA[i]]=RK[SA[i-1]]+(rk[SA[i-1]]!=rk[SA[i]]||rk[SA[i-1]+k]!=rk[SA[i]+k]);
}
void getsa(){
        p=0,q=1,k=1;
        rep(i,1,n)v[a[i]]++;
        rep(i,1,26)v[i]+=v[i-1];
        rep(i,1,n)sa[p][v[a[i]]--]=i;
        rep(i,1,n)
                rk[p][sa[p][i]]=rk[p][sa[p][i-1]]+(a[sa[p][i]]!=a[sa[p][i-1]]);
        for(k=1;k<n;k<<=1,swap(p,q))
                calsa(sa[p],rk[p],sa[q],rk[q]);
}
void geth(){
        k=0;
        rep(i,1,n)
                if(rk[p][i]==1)h[rk[p][i]]=0;
                else{
                        int j=sa[p][rk[p][i]-1];
                        while(a[i+k]==a[j+k])k++;
                        h[rk[p][i]]=k;
                        if(k>0)k--;
                }
}
int main(){
        while(T--){
                init();
                scanf("%s",str+1);
                n = strlen(str+1);
                rep(i,1,n)a[i]=str[i]-'a'+1;
                getsa();
                geth();
                h[0] = h[1] = 0;h[n+1] = 0;
        }
        return 0;
}
```

后缀自动机

```
/*
	init() 初始化
	ins(w) 从后插入新点
	getsz() 做出 parent 树，求出 right 集合大小 =sz
*/
struct SAM{
	static const int K = 26;
	int rt, la, nodes;
	int len[N], n[N][K], pa[N], sz[N];
	void init(){
		nodes = 0;
		rt = la = newnode(0);
	}
	int newnode(int pl){
		int i = ++nodes;
		len[i] = pl;
		return i;
	}
	void ins(int w){
		int p = la, np = newnode(len[p]+1);
		la = np;
		sz[np] = 1;
		while(p && !n[p][w])n[p][w] = np, p = pa[p];
		if(!p)pa[np] = rt;
		else{
			int q = n[p][w];
			if(len[q] == len[p]+1)pa[np] = q;
			else{
				int nq = newnode(len[p]+1);
				memcpy(n[nq], n[q], sizeof(n[q]));
				pa[nq] = pa[q];
				pa[q] = pa[np] = nq;
				while(p && n[p][w] == q)n[p][w] = nq, p = pa[p];
			}
		}
	}
	void getsz(){
		rep(i,2,nodes)
			adde(pa[i],i);
		dfs(rt);
	}
	void dfs(int u){
		for(int i = point[u];i;i=G[i].n){
			int v = G[i].v;
			dfs(v);
			sz[u] += sz[v];
		}
	}
}sam;
```

回文自动机

```
/*
	用法类似 sam
	本质不同的回文串有 O(n) 个
	回文树有两个根
	a 向 b 有一个 c 的转移表示对 a 表示的回文串两端都加上 c 变成 b
	分别为 even,odd, 长度分别是 0 和-1
	len 为一个点代表的字符串的实际长度
	suffix 为这个点失配后的最长回文后缀，且下标比 i 小
	n 是自动机的边
	cnt 是出现次数，向 suffix 传递，需要调用 calc()
*/
struct PAM{
	char str[N];
	int n[N][M], suffix[N], len[N], cnt[N];
	int tot, suf;
	int newnode(){
		int i = tot++;
		SET(n[i],0);
		suffix[i] = len[i] = cnt[i] = 0;
		return i;
	}
```

```
	void init(){
		tot = 0;
		int p = newnode(), q = newnode();
		len[p] = 0;
		suffix[p] = q;
		len[q] = -1;
		suffix[q] = q;
		suf = 0;
	}
	int getfail(int x, int l){
		while(str[l-1-len[x]] != str[l])
			x = suffix[x];
		return x;
	}
	int insert(int x){
		int c = str[x]-'a';
		int p = getfail(suf,x);
		if(!n[p][c]){
			int q = newnode();
			len[q] = len[p]+2;
			suffix[q] = n[getfail(suffix[p],x)][c];
			n[p][c] = q;
		}
		p = n[p][c];
		cnt[p]++;
		suf = p;
		return suf;
	}
	void calc(){
		repr(i,0,tot-1)
			cnt[suffix[i]] += cnt[i];
	}
	void debug(){
		rep(i,0,tot-1){
			pr(i),sp,pr(suffix[i]),sp,pr(cnt[i]),ln;
			rep(j,0,M-1)if(n[i][j])putchar('a'+j),sp,pr(n[i][j]),ln;
		}
	}
	void solve(){
		init();
		cin>>str;
		rep(i,0,strlen(str)-1)
			insert(i);
	}
};
```

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;
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

```
/*
        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

```
// sc(x) pr(x)
#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 int sc(int &x){
        char ch;
        int sgn = 1;
        while(blank(ch = nc()));
        if(IOerror)
                return -1;
        if(ch=='-')sgn=-1,ch=nc();
        for(x = ch - '0'; (ch = nc()) >= '0' && ch <= '9'; x = x * 10 + ch - '0');
        x*=sgn;
        return 1;
}
inline void pr(int x){
        if (x == 0){
                putchar('0');
                return;
        }
        short i, d[19];
        for (i = 0;x; ++i)
                d[i] = x % 10, x /= 10;
        while (i--)
                putchar(d[i] + '0');
}
#undef BUF_SIZE
```

高精度

```
const int base = 1000000000;
const int base_digits = 9;
struct bigint{
        vector<int> a;
        int sign; // 符号位 1 / -1
        // 基本函数
        bigint() : sign(1){}
        bigint(long long v){
                *this = v;
        }
        bigint(const string &s){
                read(s);
        }
        void operator=(const bigint &v){
                sign = v.sign;
                a = v.a;
        }
        void operator=(long long v){
                sign = 1;
                if (v < 0) sign = -1, v = -v;
                a.clear();
                for (; v > 0; v = v / base)
                        a.push_back(v % base);
        }
        // 长度
        int size(){
                if (a.empty())
                        return 0;
                int ans = (a.size() - 1) * base_digits;
                int ca = a.back();
                while (ca)
                        ans++, ca /= 10;
                return ans;
        }
        // 去前导零
        void trim(){
                while (!a.empty() && !a.back())
                        a.pop_back();
                if (a.empty())
                        sign = 1;
        }
        bool isZero() const{
                return a.empty() || (a.size() == 1 && !a[0]);
        }
```

```
        // 负号
        bigint operator-() const{
                bigint res = *this;
                res.sign = -sign;
                return res;
        }
        // 绝对值
        bigint abs() const{
                bigint res = *this;
                res.sign *= res.sign;
                return res;
        }
        // 转 long long
        long long longValue() const{
                long long res = 0;
                for (int i = a.size() - 1; i >= 0; i--)
                        res = res * base + a[i];
                return res * sign;
        }
        // 基本运算
        // 幂
        bigint operator ^(const bigint &v){
                bigint ans = 1, a = *this, b = v;
                while (!b.isZero()){
                        if (b % 2)
                                ans *= a;
                        a *= a, b /= 2;
                }
                return ans;
        }
        // 高精度加
        bigint operator+(const bigint &v) const{
                if (sign == v.sign){
                        bigint res = v;
                        for (int i = 0, carry = 0; i < (int) max(a.size(), v.a.size()) || carry; ++i){
                                if (i == (int) res.a.size())
                                        res.a.push_back(0);
                                res.a[i] += carry + (i < (int) a.size() ? a[i] : 0);
                                carry = res.a[i] >= base;
                                if (carry)
                                        res.a[i] -= base;
                        }
                        return res;
                }
                return *this - (-v);
        }
        // 高精度减
        bigint operator-(const bigint &v) const{
                if (sign == v.sign){
                        if (abs() >= v.abs()){
                                bigint res = *this;
                                for (int i = 0, carry = 0; i < (int) v.a.size() || carry; ++i){
                                        res.a[i] -= carry + (i < (int) v.a.size() ? v.a[i] : 0);
                                        carry = res.a[i] < 0;
                                        if (carry)
                                                res.a[i] += base;
                                }
                                res.trim();
                                return res;
                        }
                        return -(v - *this);
                }
                return *this + (-v);
        }
        // 高精度乘 前置函数
        static vector<int> convert_base(const vector<int> &a, int old_digits, int new_digits){
                vector<long long> p(max(old_digits, new_digits) + 1);
                p[0] = 1;
                for (int i = 1; i < (int) p.size(); i++)
                        p[i] = p[i - 1] * 10;
                vector<int> res;
                long long cur = 0;
                int cur_digits = 0;
                for (int i = 0; i < (int) a.size(); i++){
                        cur += a[i] * p[cur_digits];
                        cur_digits += old_digits;
                        while (cur_digits >= new_digits){
```

```
                    res.push_back(int(cur % p[new_digits]));
                    cur /= p[new_digits];
                    cur_digits -= new_digits;
                }
            }
            res.push_back((int) cur);
            while (!res.empty() && !res.back())
                res.pop_back();
            return res;
        }
        typedef vector<long long> vll;
        // 高精度乘 前置函数
        static vll karatsubaMultiply(const vll &a, const vll &b){
            int n = a.size();
            vll res(n + n);
            if (n <= 32){
                for (int i = 0; i < n; i++)
                    for (int j = 0; j < n; j++)
                        res[i + j] += a[i] * b[j];
                return res;
            }
            int k = n >> 1;
            vll a1(a.begin(), a.begin() + k);
            vll a2(a.begin() + k, a.end());
            vll b1(b.begin(), b.begin() + k);
            vll b2(b.begin() + k, b.end());
            vll a1b1 = karatsubaMultiply(a1, b1);
            vll a2b2 = karatsubaMultiply(a2, b2);
            for (int i = 0; i < k; i++)
                a2[i] += a1[i];
            for (int i = 0; i < k; i++)
                b2[i] += b1[i];
            vll r = karatsubaMultiply(a2, b2);
            for (int i = 0; i < (int) a1b1.size(); i++)
                r[i] -= a1b1[i];
            for (int i = 0; i < (int) a2b2.size(); i++)
                r[i] -= a2b2[i];
            for (int i = 0; i < (int) r.size(); i++)
                res[i + k] += r[i];
            for (int i = 0; i < (int) a1b1.size(); i++)
                res[i] += a1b1[i];
            for (int i = 0; i < (int) a2b2.size(); i++)
                res[i + n] += a2b2[i];
            return res;
        }
        // 高精度乘 需要两个前置函数
        bigint operator*(const bigint &v) const{
            vector<int> a6 = convert_base(this->a, base_digits, 6);
            vector<int> b6 = convert_base(v.a, base_digits, 6);
            vll a(a6.begin(), a6.end());
            vll b(b6.begin(), b6.end());
            while (a.size() < b.size())
                a.push_back(0);
            while (b.size() < a.size())
                b.push_back(0);
            while (a.size() & (a.size() - 1))
                a.push_back(0), b.push_back(0);
            vll c = karatsubaMultiply(a, b);
            bigint res;
            res.sign = sign * v.sign;
            for (int i = 0, carry = 0; i < (int) c.size(); i++){
                long long cur = c[i] + carry;
                res.a.push_back((int) (cur % 1000000));
                carry = (int) (cur / 1000000);
            }
            res.a = convert_base(res.a, 6, base_digits);
            res.trim();
            return res;
        }
        // 高精度除/取模 前置函数
        friend pair<bigint, bigint> divmod(const bigint &a1, const bigint &b1){
            int norm = base / (b1.a.back() + 1);
            bigint a = a1.abs() * norm;
            bigint b = b1.abs() * norm;
            bigint q, r;
            q.a.resize(a.a.size());
            for (int i = a.a.size() - 1; i >= 0; i--){
```

```
            r *= base;
            r += a.a[i];
            int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
            int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() - 1];
            int d = ((long long) base * s1 + s2) / b.a.back();
            r -= b * d;
            while (r < 0)
                r += b, --d;
            q.a[i] = d;
        }
        q.sign = a1.sign * b1.sign;
        r.sign = a1.sign;
        q.trim();
        r.trim();
        return make_pair(q, r / norm);
    }
    // 高精度除
    bigint operator/(const bigint &v) const{
        return divmod(*this, v).first;
    }
    // 高精度取模
    bigint operator%(const bigint &v) const{
        return divmod(*this, v).second;
    }
    void operator+=(const bigint &v){
        *this = *this + v;
    }
    void operator-=(const bigint &v){
        *this = *this - v;
    }
    void operator*=(const bigint &v){
        *this = *this * v;
    }
    void operator/=(const bigint &v){
        *this = *this / v;
    }
    // 低精度乘
    void operator*=(int v){
        if (v < 0)
            sign = -sign, v = -v;
        for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i){
            if (i == (int) a.size())
                a.push_back(0);
            long long cur = a[i] * (long long) v + carry;
            carry = (int) (cur / base);
            a[i] = (int) (cur % base);
        }
        trim();
    }
    // 低精度乘
    bigint operator*(int v) const{
        bigint res = *this;
        res *= v;
        return res;
    }
    // 低精度除
    void operator/=(int v){
        if (v < 0)
            sign = -sign, v = -v;
        for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i){
            long long cur = a[i] + rem * (long long) base;
            a[i] = (int) (cur / v);
            rem = (int) (cur % v);
        }
        trim();
    }
    // 低精度除
    bigint operator/(int v) const{
        bigint res = *this;
        res /= v;
        return res;
    }
    // 低精度模
    int operator%(int v) const{
        if (v < 0)
            v = -v;
        int m = 0;
```

```cpp
        for (int i = a.size() - 1; i >= 0; --i){
            m = (a[i] + m * (long long) base) % v;
        }
        return m * sign;
    }
    // 比较关系
    bool operator<(const bigint &v) const{
        if (sign != v.sign)
            return sign < v.sign;
        if (a.size() != v.a.size())
            return a.size() * sign < v.a.size() * v.sign;
        for (int i = a.size() - 1; i >= 0; i--)
            if (a[i] != v.a[i])
                return a[i] * sign < v.a[i] * sign;
        return false;
    }
    bool operator>(const bigint &v) const{
        return v < *this;
    }
    bool operator<=(const bigint &v) const{
        return !(v < *this);
    }
    bool operator>=(const bigint &v) const{
        return !(*this < v);
    }
    bool operator==(const bigint &v) const{
        return !(*this < v) && !(v < *this);
    }
    bool operator!=(const bigint &v) const{
        return *this < v || v < *this;
    }
    // 输入输出
    void read(const string &s){
        sign = 1;
        a.clear();
        int pos = 0;
        while (pos < (int) s.size() && (s[pos] == '-' || s[pos] == '+')){
            if (s[pos] == '-')
                sign = -sign;
            ++pos;
        }
        for (int i = s.size() - 1; i >= pos; i -= base_digits){
            int x = 0;
            for (int j = max(pos, i - base_digits + 1); j <= i; j++)
                x = x * 10 + s[j] - '0';
            a.push_back(x);
        }
        trim();
    }
    friend istream& operator>>(istream &stream, bigint &v){
        string s;
        stream >> s;
        v.read(s);
        return stream;
    }
    friend ostream& operator<<(ostream &stream, const bigint &v){
        if (v.sign == -1)
            stream << '-';
        stream << (v.a.empty() ? 0 : v.a.back());
        for (int i = (int) v.a.size() - 2; i >= 0; --i)
            stream << setw(base_digits) << setfill('0') << v.a[i];
        return stream;
    }
    // 扩展功能
    friend bigint gcd(const bigint &a, const bigint &b){
        return b.isZero() ? a : gcd(b, a % b);
    }
    friend bigint lcm(const bigint &a, const bigint &b){
        return a / gcd(a, b) * b;
    }
    friend bigint sqrt(const bigint &a1) {
    bigint a = a1;
    while (a.z.empty() || a.z.size() % 2 == 1)
      a.z.push_back(0);

    int n = a.z.size();

```

```
        int firstDigit = (int) sqrt((double) a.z[n - 1] * base + a.z[n - 2]);
        int norm = base / (firstDigit + 1);
        a *= norm;
        a *= norm;
        while (a.z.empty() || a.z.size() % 2 == 1)
            a.z.push_back(0);

        bigint r = (long long) a.z[n - 1] * base + a.z[n - 2];
        firstDigit = (int) sqrt((double) a.z[n - 1] * base + a.z[n - 2]);
        int q = firstDigit;
        bigint res;

        for(int j = n / 2 - 1; j >= 0; j--) {
            for(; ; --q) {
                bigint r1 = (r - (res * 2 * base + q) * q) * base * base + (j > 0 ? (long long) a.z[2 * j - 1] * base + a.z[2 * j - 2] : 0);
                if (r1 >= 0) {
                    r = r1;
                    break;
                }
            }
            res *= base;
            res += q;

            if (j > 0) {
                int d1 = res.z.size() + 2 < r.z.size() ? r.z[res.z.size() + 2] : 0;
                int d2 = res.z.size() + 1 < r.z.size() ? r.z[res.z.size() + 1] : 0;
                int d3 = res.z.size() < r.z.size() ? r.z[res.z.size()] : 0;
                q = ((long long) d1 * base * base + (long long) d2 * base + d3) / (firstDigit * 2);
            }
        }

        res.trim();
        return res / norm;
    }
};
```

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;
}
```

魔法求递推式

```
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define fi first
#define se second
#define SZ(x) ((int)(x).size())
typedef vector<int> VI;
typedef long long ll;
typedef pair<int,int> PII;
```

```cpp
const ll mod=1000000007;
ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0); for(;b;b>>=1){if(b&1)res=res*a%mod;a=a*a%mod;}return res;}
// head
int _;
ll n;
namespace linear_seq {
    const int N=10010;
    ll res[N],base[N],_c[N],_md[N];
    vector<int> Md;
    void mul(ll *a,ll *b,int k) {
        rep(i,0,k+k) _c[i]=0;
        rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
        for (int i=k+k-1;i>=k;i--) if (_c[i])
            rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[j]]-_c[i]*_md[Md[j]])%mod;
        rep(i,0,k) a[i]=_c[i];
    }
    int solve(ll n,VI a,VI b) { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
        ll ans=0,pnt=0;
        int k=SZ(a);
        assert(SZ(a)==SZ(b));
        rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
        Md.clear();
        rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
        rep(i,0,k) res[i]=base[i]=0;
        res[0]=1;
        while ((1ll<<pnt)<=n) pnt++;
        for (int p=pnt;p>=0;p--) {
            mul(res,res,k);
            if ((n>>p)&1) {
                for (int i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;
                rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%mod;
            }
        }
        rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
        if (ans<0) ans+=mod;
        return ans;
    }
    VI BM(VI s) {
        VI C(1,1),B(1,1);
        int L=0,m=1,b=1;
        rep(n,0,SZ(s)) {
            ll d=0;
            rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
            if (d==0) ++m;
            else if (2*L<=n) {
                VI T=C;
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                L=n+1-L; B=T; b=d; m=1;
            } else {
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                ++m;
            }
        }
        return C;
    }
    int gao(VI a,ll n) {
        VI c=BM(a);
        c.erase(c.begin());
        rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
        return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
    }
};
int main() {
    for (scanf("%d",&_);_;_--) {
        scanf("%lld",&n);
        printf("%d\n",linear_seq::gao(VI{x1,x2,x3,x4},n-1));
    }
}
```

常用概念

欧拉路径

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

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

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

pick 定理

整多边形面积 $A=$ 内部格点数 $i+$ 边上格点数 $\frac{b}{2} - 1$

重心

半径为 r , 圆心角为 θ 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 θ 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

Lucas 定理

$C(n,m)modp = C(nmodp, mmodp) * C(n/p, m/p), p$ 是质数

扩展 Lucas 定理

若 p 不是质数，将 p 分解质因数后分别求解，再用中国剩余定理合并

BEST theorem

有向图中欧拉回路的数量 $\mathrm{ec}(G) = t_w(G) \prod_{v \in V} \big(\deg(v) - 1\big)!.$

其中 $deg(v)$ 表示 v 的入度，$tw(G)$ 表示以 w 为根的外向树的数量，且在连通欧拉图中以任一点为根的外向树数量相同

若需要定起点，则答案乘上 $deg(s)$，表示对每一条欧拉回路，s 出现了 $deg(s)$ 次，选取一个点切开得到一条从 s 出发的欧拉回路

欧拉示性数定理

对平面图 $V - E + F = 2$

Polya 定理

设对 n 个对象用 m 种颜色：$b_1, b_2, \cdots, b_m$ 着色。

设 $m^{c(p_i)} = (b_1 + b_2 + \cdots + b_m)^{c_1(p_i)}(b_1^2 + b_2^2 + \cdots + b_m^2)^{c_2(p_i)} \cdots (b_1^n + b_2^n + \cdots + b_m^n)^{c_n(p_i)}$，其中 $c_j(p_i)$ 表示置换群中第 i 个置换循环长度为 j 的个数。

设 $S_k = (b_1^k + b_2^k + \cdots + b_m^k), k = 1, 2 \cdots, n$，则波利亚计数定理的母函数形式为：$P(G) = \dfrac{1}{\mid G \mid} \sum_{j=1}^{g} \Pi_{k=1}^{n} S_k^{c_k(p_j)}$

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

常用排列组合公式

$\sum_{i=1}^{n} x_i = k, x_i \geq 0$ 的解数为 $C(n+k-1, n-1)$
$x_1 \geq 0, x_i \leq x_{i+1}, x_n \leq k-1$ 的解数等价于在 [0,k-1] 共 k 个数中可重复的取 n 个数的组合数，为 $C(n+k-1, n)$

三角公式

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

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

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

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

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

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

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

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

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

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

积分表

== 含有 $ax+b$ 的积分 ==

$\int (ax+b)^n \mathrm{d}x = \frac{(ax+b)^{n+1}}{a(n+1)} + C$

$\int \frac{1}{ax+b}\mathrm{d}x = \frac{1}{a}\ln|ax+b| + C$

$\int \frac{x}{ax+b}\mathrm{d}x = \frac{1}{a^2}(ax+b-b\ln|ax+b|) + C$

$\int \frac{x^2}{ax+b}\mathrm{d}x = \frac{1}{2a^3}\left[(ax+b)^2 - 4b(ax+b) + 2b^2\ln|ax+b|\right] + C$

$\int \frac{1}{x(ax+b)}\mathrm{d}x = -\frac{1}{b}\ln\left|\frac{ax+b}{x}\right| + C$

$\int \frac{1}{x^2(ax+b)}\mathrm{d}x = \frac{a}{b^2}\ln\left|\frac{ax+b}{x}\right| - \frac{1}{bx} + C$

== 含有 $\sqrt{a+bx}$ 的积分 ==

$\int x\sqrt{a+bx}\mathrm{d}x = \frac{2}{15b^2}(3bx-2a)(a+bx)^{\frac{3}{2}} + C$

$\int x^2\sqrt{a+bx}\mathrm{d}x = \frac{2}{105b^3}(15b^2x^2 - 12abx + 8a^2)(a+bx)^{\frac{3}{2}} + C$

$\int x^n\sqrt{a+bx}\mathrm{d}x = \frac{2}{b(2n+3)}x^n(a+bx)^{\frac{3}{2}} - \frac{2na}{b(2n+3)}\int x^{n-1}\sqrt{a+bx}\mathrm{d}x$

$\int \frac{\sqrt{a+bx}}{x}\mathrm{d}x = 2\sqrt{a+bx} + a\int \frac{1}{x\sqrt{a+bx}}\mathrm{d}x$

$\int \frac{\sqrt{a+bx}}{x^n}\mathrm{d}x = \frac{-1}{a(n-1)}\frac{(a+bx)^{\frac{3}{2}}}{x^{n-1}} - \frac{(2n-5)b}{2a(n-1)}\int \frac{\sqrt{a+bx}}{x^{n-1}}\mathrm{d}x, n \neq 1$

$\int \frac{1}{x\sqrt{a+bx}}\mathrm{d}x = \frac{1}{\sqrt{a}}\ln\left(\frac{\sqrt{a+bx}-\sqrt{a}}{\sqrt{a+bx}+\sqrt{a}}\right) + C, a > 0 = \frac{2}{\sqrt{-a}}\arctan\sqrt{\frac{a+bx}{-a}} + C, a < 0$

$\int \frac{1}{x^n\sqrt{a+bx}}\mathrm{d}x = \frac{-1}{a(n-1)}\frac{\sqrt{a+bx}}{x^{n-1}} - \frac{(2n-3)b}{2a(n-1)}\int \frac{1}{x^{n-1}}\sqrt{a+bx}\mathrm{d}x, n \neq 1$

== 含有 $x^2 \pm \alpha^2$ 的积分 ==

$\int \frac{1}{x^2+\alpha^2}\mathrm{d}x = \frac{\arctan\frac{x}{\alpha}}{\alpha} + C$

$\int \frac{1}{\pm x^2 \mp \alpha^2}\mathrm{d}x = \frac{\ln\left(\frac{x \mp \alpha}{\pm x + \alpha}\right)}{2\alpha} + C$

== 含有 ax^2+b 的积分 ==

$\int \frac{1}{ax^2+b}\mathrm{d}x = \frac{1}{\sqrt{ab}}\arctan\frac{\sqrt{a}x}{\sqrt{b}} + C$

== 含有 $ax^2+bx+c \qquad (a > 0)$ 的积分 ==

$\int ax^2 + bx + c\mathrm{d}x = \frac{ax^3}{3} + \frac{bx^2}{2} + cx + C$

== 含有 $\sqrt{a^2+x^2} \qquad (a > 0)$ 的积分 ==

$\int \sqrt{a^2+x^2}\mathrm{d}x = \frac{1}{2}x\sqrt{a^2+x^2} + \frac{1}{2}a^2\ln\left(x + \sqrt{a^2+x^2}\right) + C$

$\int x^2\sqrt{a^2+x^2}\mathrm{d}x = \frac{1}{8}x(a^2+2x^2)\sqrt{a^2+x^2} - \frac{1}{8}a^4\ln\left(x+\sqrt{a^2+x^2}\right) + C$

$\int \frac{\sqrt{a^2+x^2}}{x}\mathrm{d}x = \sqrt{a^2+x^2} - a\ln\left(\frac{a+\sqrt{a^2+x^2}}{x}\right) + C$

$\int \frac{\sqrt{a^2+x^2}}{x^2}\mathrm{d}x = \ln\left(x+\sqrt{a^2+x^2}\right) - \frac{\sqrt{a^2+x^2}}{x} + C$

$\int \frac{1}{\sqrt{a^2+x^2}}\mathrm{d}x = \ln\left(x+\sqrt{a^2+x^2}\right) + C$

$\int \frac{x^2}{\sqrt{a^2+x^2}}\mathrm{d}x = \frac{1}{2}x\sqrt{a^2+x^2} - \frac{1}{2}a^2\ln\left(\sqrt{a^2+x^2}+x\right) + C$

$\int \frac{1}{x\sqrt{a^2+x^2}}\mathrm{d}x = \frac{1}{a}\ln\left(\frac{x}{a+\sqrt{a^2+x^2}}\right) + C$

$\int \frac{1}{x^2\sqrt{a^2+x^2}}\mathrm{d}x = -\frac{\sqrt{a^2+x^2}}{a^2x} + C$

== 含有 $\sqrt{x^2-a^2} \qquad (x^2 > a^2)$ 的积分 =

$\int \frac{1}{\sqrt{x^2-a^2}}\mathrm{d}x = \ln|x+\sqrt{x^2-a^2}|+C$

== 含有 $\sqrt{a^2-x^2} \qquad (a^2>x^2)$ 的积分 ==

$\int \frac{1}{\sqrt{a^2-x^2}}\mathrm{d}x = \arcsin\frac{x}{a}+C = -\arccos\frac{x}{a}+C$

$\int \sqrt{a^2-x^2}\mathrm{d}x = \frac{1}{2}x\sqrt{a^2-x^2}+\frac{a^2}{2}\arcsin\frac{x}{a}+C$

$\int x^2\sqrt{a^2-x^2}\mathrm{d}x = \frac{1}{8}x(2x^2-a^2)\sqrt{a^2-x^2}+\frac{1}{8}a^4\arcsin\frac{x}{a}+C$

$\int \frac{\sqrt{a^2-x^2}}{x}\mathrm{d}x = \sqrt{a^2-x^2}-a\ln\left(\frac{a+\sqrt{a^2-x^2}}{x}\right)+C$

$\int \frac{\sqrt{a^2-x^2}}{x^2}\mathrm{d}x = -\frac{\sqrt{a^2-x^2}}{x}-\arcsin\frac{x}{a}+C$

$\int \frac{1}{x\sqrt{a^2-x^2}}\mathrm{d}x = -\frac{1}{a}\ln\left(\frac{a+\sqrt{a^2-x^2}}{x}\right)+C$

$\int \frac{x^2}{\sqrt{a^2-x^2}}\mathrm{d}x = -\frac{1}{2}x\sqrt{a^2-x^2}+\frac{1}{2}a^2\arcsin\frac{x}{a}+C$

$\int \frac{1}{x^2\sqrt{a^2-x^2}}\mathrm{d}x = -\frac{\sqrt{a^2-x^2}}{a^2x}+C$

== 含有 $R=\sqrt{|a|x^2+bx+c} \qquad (a\neq 0)$ 的积分 ==

$\int \frac{\mathrm{d}x}{R} = \frac{1}{\sqrt{a}}\ln\left(2\sqrt{a}R+2ax+b\right) \qquad (\text{for } a>0)$

$\int \frac{\mathrm{d}x}{R} = \frac{1}{\sqrt{a}}\ \operatorname{arsinh}\frac{2ax+b}{\sqrt{4ac-b^2}} \qquad (\text{for } a>0,\ 4ac-b^2>0)$

$\int \frac{\mathrm{d}x}{R} = \frac{1}{\sqrt{a}}\ln|2ax+b| \quad (\text{for } a>0,\ 4ac-b^2=0)$

$\int \frac{\mathrm{d}x}{R} = -\frac{1}{\sqrt{-a}}\arcsin\frac{2ax+b}{\sqrt{b^2-4ac}} \qquad (\text{for } a<0,\ 4ac-b^2<0,\ (2ax+b)<\sqrt{b^2-4ac})$

$\int \frac{\mathrm{d}x}{R^3} = \frac{4ax+2b}{(4ac-b^2)R}$

$\int \frac{\mathrm{d}x}{R^5} = \frac{4ax+2b}{3(4ac-b^2)R}\left(\frac{1}{R^2}+\frac{8a}{4ac-b^2}\right)$

$\int \frac{\mathrm{d}x}{R^{2n+1}} = \frac{2}{(2n-1)(4ac-b^2)}\left[\frac{2ax+b}{R^{2n-1}}+4a(n-1)\int\frac{\mathrm{d}x}{R^{2n-1}}\right]$

$\int \frac{x}{R}\ \mathrm{d}x = \frac{R}{a}-\frac{b}{2a}\int\frac{\mathrm{d}x}{R}$

$\int \frac{x}{R^3}\ \mathrm{d}x = -\frac{2bx+4c}{(4ac-b^2)R}$

$\int \frac{x}{R^{2n+1}}\ \mathrm{d}x = -\frac{1}{(2n-1)aR^{2n-1}}-\frac{b}{2a}\int\frac{\mathrm{d}x}{R^{2n+1}}$

$\int \frac{\mathrm{d}x}{xR} = -\frac{1}{\sqrt{c}}\ln\left(\frac{2\sqrt{c}R+bx+2c}{x}\right)$

$\int \frac{\mathrm{d}x}{xR} = -\frac{1}{\sqrt{c}}\operatorname{arsinh}\left(\frac{bx+2c}{|x|\sqrt{4ac-b^2}}\right)$

== 含有三角函数的积分 ==

$\int \cos x\mathrm{d}x = \sin x+C$

$\int \sin x\mathrm{d}x = -\cos x+C$

$\int \sec^2 x\mathrm{d}x = \tan x+C$

$\int \csc^2 x\mathrm{d}x = -\cot x+C$

$\int \sec x\tan x\mathrm{d}x = \sec x+C$

$\int \csc x\cot x\mathrm{d}x = -\csc x+C$

$\int \tan x\mathrm{d}x = -\ln|\cos x|+C = \ln|\sec x|+C$

$\int \cot x\mathrm{d}x = \ln|\sin x|+C$

$\int \sec x\mathrm{d}x = \ln|\sec x+\tan x|+C$

$\int \csc x\mathrm{d}x = -\ln|\csc x+\cot x|+C = \ln\left|\frac{\tan x-\sin x}{\sin x\tan x}\right|+C$

$\int \sin^n x\mathrm{d}x = -\frac{1}{n}\sin^{n-1}x\cos x+\frac{n-1}{n}\int\sin^{n-2}x\mathrm{d}x+C \quad \forall n\geq 2$

$\int \sin^2 x\mathrm{d}x = \frac{x}{2}-\frac{\sin 2x}{4}+C$

$\int \cos^n x\mathrm{d}x = \frac{1}{n}\cos^{n-1}x\sin x+\frac{n-1}{n}\int\cos^{n-2}x\mathrm{d}x+C \quad \forall n\geq 2$

$\int \cos^2 x\mathrm{d}x = \frac{x}{2}+\frac{\sin 2x}{4}+C$

$\int \tan^n x\mathrm{d}x = \frac{1}{n-1}\tan^{n-1}x-\int\tan^{n-2}x\mathrm{d}x+C \quad \forall n\geq 2$

$\int \tan^2 x\mathrm{d}x = \tan x-x+C$

$\int \cot^n x\mathrm{d}x = \frac{1}{n-1}\cot^{n-1}x-\int\cot^{n-2}x\mathrm{d}x+C \quad \forall n\geq 2$

$\int \cot^2 x\mathrm{d}x = -\cot x-x+C$

$\int \sec^n x\mathrm{d}x = \frac{1}{n-1}\sec^{n-2}x\tan x+\frac{n-2}{n-1}\int\sec^{n-2}x\mathrm{d}x+C \quad \forall n\geq 2$

$\int \csc^n x\mathrm{d}x = -\frac{1}{n-1}\csc^{n-2}x\cot x+\frac{n-2}{n-1}\int\csc^{n-2}x\mathrm{d}x+C \quad \forall n\geq 2$

== 含有反三角函数的积分 ==

$\int \arcsin x\mathrm{d}x = x\arcsin x+\sqrt{1-x^2}+C$

$\int \arccos x \mathrm{d}x = x \arccos x - \sqrt{1-x^2} + C$

$\int \arctan x \mathrm{d}x = x \arctan x - \ln \sqrt{1+x^2} + C$

$\int arccot(x) \mathrm{d}x = x \times arccot(x) + \ln \sqrt{1+x^2} + C$

$\int arcsec(x) \mathrm{d}x = x \times arcsec(x) - sgn(x) \ln \left| x + \sqrt{x^2-1} \right| + C = x \times arcsec(x) + sgn(x) \ln \left| x - \sqrt{x^2-1} \right| + C$

$\int arccsc(x) \mathrm{d}x = x \times arccsc(x) + sgn(x) \ln \left| x + \sqrt{x^2-1} \right| + C = x \times arccsc(x) - sgn(x) \ln \left| x - \sqrt{x^2-1} \right| + C$

== 含有指数函数的积分 ==

$\int e^x \mathrm{d}x = e^x + C$

$\int \alpha^x \mathrm{d}x = \frac{\alpha^x}{\ln \alpha} + C$

$\int x e^{ax} \mathrm{d}x = \frac{1}{a^2}(ax-1)e^{ax} + C$

$\int x^n e^{ax} \mathrm{d}x = \frac{1}{a} x^n e^{ax} - \frac{n}{a} \int x^{n-1} e^{ax} \mathrm{d}x$

$\int e^{ax} \sin bx \mathrm{d}x = \frac{e^{ax}}{a^2+b^2}(a \sin bx - b \cos bx) + C$

$\int e^{ax} \cos bx \mathrm{d}x = \frac{e^{ax}}{a^2+b^2}(a \cos bx + b \sin bx) + C$

== 含有对数函数的积分 ==

$\int \ln x \mathrm{d}x = x \ln x - x + C$

$\int \log_\alpha x \mathrm{d}x = \frac{1}{\ln \alpha}(x \ln x - x) + C$

$\int x^n \ln x \mathrm{d}x = \frac{x^{n+1}}{(n+1)^2}[(n+1) \ln x - 1] + C$

$\int \frac{1}{x \ln x} \mathrm{d}x = \ln(\ln x) + C$

== 含有双曲函数的积分 ==

$\int \sinh x \mathrm{d}x = \cosh x + C$

$\int \cosh x \mathrm{d}x = \sinh x + C$

$\int \tanh x \mathrm{d}x = \ln(\cosh x) + C$

$\int \coth x \mathrm{d}x = \ln(\sinh x) + C$

$\int \mathrm{sech}\ x \mathrm{d}x = \arcsin(\tanh x) + C = \arctan(\sinh x) + C$

$\int \mathrm{csch}\ x \mathrm{d}x = \ln\left(\tanh \frac{x}{2}\right) + C$

== 定积分 ==

$\int_{-\infty}^{\infty} e^{-\alpha x^2} \mathrm{d}x = \sqrt{\frac{\pi}{\alpha}}$

$\int_0^{\frac{\pi}{2}} \sin^n x \mathrm{d}x = \int_0^{\frac{\pi}{2}} \cos^n x \mathrm{d}x =$

$$\begin{cases} \frac{n-1}{n} \cdot \frac{n-3}{n-2} \cdot \ldots \cdot \frac{4}{5} \cdot \frac{2}{3}, & \text{if } n > 1 \text{ 且} n \text{ 为奇数} \\ \frac{n-1}{n} \cdot \frac{n-3}{n-2} \cdot \ldots \cdot \frac{3}{4} \cdot \frac{1}{2} \cdot \frac{\pi}{2}, & \text{if } n > 0 \text{ 且} n \text{ 为偶数} \end{cases}$$