ACM 模板

yijiull

2017-11-3

数据范围：

unsigned int 0 ~ 4 294 967 295
int 2147483648 ~ 2 147 483 647
unsigned long 0 ~ 4 294 967 295
long 2 147 483 648 ~ 2 147 483 647
long long 的最大值：9 223 372 036 854 775 807
long long 的最小值：-9 223 372 036 854 775 808
unsigned long long 的最大值：18 446 744 073 709 551 615

__int64 的最大值：9 223 372 036 854 775 807
__int64 的最小值：-9 223 372 036 854 775 808
unsigned __int64 的最大值：18 446 744 073 709 551 615

计算几何

atan2(0,0)=0,
atan2(1,0)=pi/2,
atan2(-1,0)=-pi/2,
atan2(0,1)=0,

atan2(0,-1)=pi.

几何公式：

三角形:

1. 半周长 P=(a+b+c)/2
2. 面积 S=aHa/2=absin(C)/2=sqrt(P(P-a)(P-b)(P-c))
3. 中线 Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2
4. 角平分线 Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)
5. 高线 Ha=bsin(C)=csin(B)=sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)
6. 内切圆半径 r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)
=4Rsin(A/2)sin(B/2)sin(C/2)=sqrt((P-a)(P-b)(P-c)/P)
=Ptan(A/2)tan(B/2)tan(C/2)
7. 外接圆半径 R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C))

四边形:

D1,D2 为对角线,M 对角线中点连线,A 为对角线夹角

1. a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2
2. S=D1D2sin(A)/2 (以下对圆的内接四边形)
3. ac+bd=D1D2
4. S=sqrt((P-a)(P-b)(P-c)(P-d)),P 为半周长

正 n 边形:

R 为外接圆半径,r 为内切圆半径

1. 中心角 A=2PI/n
2. 内角 C=(n-2)PI/n
3. 边长 a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)
4. 面积 S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))

圆:

1. 弧长 l=rA
2. 弦长 a=2sqrt(2hr-h^2)=2rsin(A/2)
3. 弓形高 h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2
4. 扇形面积 S1=rl/2=r^2A/2
5. 弓形面积 S2=(rl-a(r-h))/2=r^2(A-sin(A))/2

棱柱:

1. 体积 V=Ah,A 为底面积,h 为高
2. 侧面积 S=lp,l 为棱长,p 为直截面周长
3. 全面积 T=S+2A

棱锥:

1. 体积 V=Ah/3,A 为底面积,h 为高（以下对正棱锥）
2. 侧面积 S=lp/2,l 为斜高,p 为底面周长
3. 全面积 T=S+A

棱台:
1. 体积 V=(A1+A2+sqrt(A1A2))h/3,A1.A2 为上下底面积,h 为高（以下为正棱台）
2. 侧面积 S=(p1+p2)l/2,p1.p2 为上下底面周长,l 为斜高
3. 全面积 T=S+A1+A2

圆柱:
1. 侧面积 S=2Plrh
2. 全面积 T=2Plr(h+r)
3. 体积 V=Plr^2h

圆锥:
1. 母线 l=sqrt(h^2+r^2)
2. 侧面积 S=Plrl
3. 全面积 T=Plr(l+r)
4. 体积 V=Plr^2h/3

圆台:
1. 母线 l=sqrt(h^2+(r1-r2)^2)
2. 侧面积 S=Pl(r1+r2)l
3. 全面积 T=Plr1(l+r1)+Plr2(l+r2)
4. 体积 V=Pl(r1^2+r2^2+r1r2)h/3

球:
1. 全面积 T=4Plr^2
2. 体积 V=4Plr^3/3

球台:
1. 侧面积 S=2Plrh
2. 全面积 T=Pl(2rh+r1^2+r2^2)
3. 体积 V=Plh(3(r1^2+r2^2)+h^2)/6

球扇形:
1. 全面积 T=Plr(2h+r0),h 为球冠高,r0 为球冠底面半径
2. 体积 V=2Plr^2h/3

皮克定理：

计算点阵中顶点在格点上的多边形面积公式：S=a+b/2-1（为避免精度问题，2S = 2a + b − 2）

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

```
void pick(Point *p, int n, LL &in, LL &on){
    LL area = 0;
    on = 0;
    p[n] = p[0];
    for(int i = 0; i < n; i++) {
        area += cross(p[i], p[i+1]);
        on += gcd(abs(p[i].x - p[i+1].x), abs(p[i].y - p[i+1].y));
    }
    area = abs(area);
    in = (area + 2 - on) / 2;
}
```

二维

点&&向量

```
struct Point {
    double x,y;
    Point (double x = 0, double y = 0) : x(x), y(y) {}
};
typedef Point Vector;
Vector operator + (Vector a, Vector b) {
    return Vector (a.x + b.x, a.y + b.y);
}
Vector operator * (Vector a, double s) {
    return Vector (a.x * s, a.y * s);
}
Vector operator / (Vector a, double p) {
    return Vector (a.x / p, a.y / p);
}
Vector operator - (Point a, Point b) {
    return Vector (a.x - b.x, a.y - b.y);
}
bool operator < (Point a, Point b) {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}
int dcmp (double x) {
    if(fabs(x) < eps) return 0;
    return x < 0 ? -1 : 1;
}
bool operator == (const Point &a, const Point &b) {
    return dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y) == 0;
```

```
}
```

基础运算

```
double Angel (Vector a) {
    return atan2(a.y, a.x);
}
double Dot(Vector a, Vector b) {
    return a.x * b.x + a.y * b.y;
}
double Length (Vector a) {
    return sqrt(Dot(a, a));
}
double Angle (Vector a, Vector b) {
    return acos(Dot(a, b) / Length(a) / Length(b));
}
double Cross (Vector a, Vector b) {
    return a.x * b.y - a.y * b.x;
}
double   Area2 (Point a, Point b, Point c) {
    return Cross(b - a, c - a);
}
Vector Rotate (Vector a, double rad) {   //右手逆时针旋转 rad(弧度)
    return Vector (a.x * cos(rad) - a.y * sin(rad), a.x * sin(rad) + a.y * cos(rad));
}
Vector Normal (Vector a) {
    double L = Length(a);
    return Vector(-a.y / L, a.x / L);
}
```

简单常用函数

```
//两直线交点
Point GetLineIntersection (Point p, Vector v, Point q, Vector w) {
    Vector u = p - q;
    double t1 = Cross(w, u) / Cross(v, w);
    double t2 = Cross(v, u) / Cross(v, w);
    return p + v * t1;   //   return q + w * t2;
}
//点到直线的 dis
double DistanceToLine(Point p, Point a, Point b) {
```

```
    Vector v1 = b - a, v2 = p - a;
    return fabs(Cross(v1, v2)) / Length(v1);
}
//点到线段的 dis
double DistanceToSegment(Point p, Point a, Point b) {
    if(a == b) return Length(a - p);
    Vector v1 = b - a, v2 = p - a, v3 = p - b;
    if(dcmp(Dot(v1, v2)) < 0) return Length(v2);
    else if(dcmp(Dot(v1, v3)) > 0) return Length(v3);
    else return fabs(Cross(v1, v2)) / Length(v1);
}
//两线段的距离
double SegmentDistance(Point a1, Point b1, Point a2, Point b2) {
    return min(min(DistanceToSegment(a1, a2, b2), DistanceToSegment(b1, a2, b2)),
                    min(DistanceToSegment(a2, a1, b1), DistanceToSegment(b2, a1, b1)));
}
//点在直线的投影
Point GetLineProjection(Point p, Point a, Point b) {
    Vector v = b - a;
    return a + v * (Dot(v, p-a) / Dot(v, v));
}
//判断两线段是否规范相交
bool SegmentProperIntersection(Point a1, Point b1, Point a2, Point b2) {
    double c1 = Cross(b1 - a1, a2 - a1), c2 = Cross(b1 - a1, b2 - a1),
           c3 = Cross(b2 - a2, a1 - a2), c4 = Cross(b2 - a2, b1 - a2);
    return dcmp(c1)*dcmp(c2) < 0 && dcmp(c3)*dcmp(c4) < 0;
}
//判断点是否在线段上(即∠APB 等于 π)
bool OnSegment(Point p, Point a, Point b) {
    return dcmp(Cross(a - p, b - p)) == 0 && dcmp(Dot(a - p, b - p)) < 0;
}
//多边形面积
double ConvexPolygonArea(Point *p, int n) {
    double area = 0;
    for(int i = 1; i < n-1; i++) {
        area += Cross(p[i] - p[0], p[i+1] - p[0]);
    }
    return area / 2;
}
//判断点是否在多边形内 or 外 or 上
bool isPointInPolygon(Point p,Point *poly, int n) {
    int cnt = 0;
    poly[n] = poly[0];
    for(int i = 0; i < n; i++) {
```

```
            if(OnSegment(p, poly[i], poly[i+1])) return 0;    //在边界上
            int k = dcmp(Cross(poly[i+1]-poly[i], p - poly[i]));
            int d1 = dcmp(poly[i].y - p.y);
            int d2 = dcmp(poly[i+1].y - p.y);
            if(k > 0 && d1 <= 0 && d2 > 0) cnt++;
            if(k < 0 && d2 <= 0 && d1 >= 0) cnt--;
        }
        if(cnt) return -1; //内部
        return 1; //外部
}
```

切割多边形

```
// cut with directed line A->B, return the left part
// may return a single point or a line segment
Polygon CutPolygon(Polygon poly, Point A, Point B) {
   Polygon newpoly;
   int n = poly.size();
   for(int i = 0; i < n; i++) {
      Point C = poly[i];
      Point D = poly[(i+1)%n];
      if(dcmp(Cross(B-A, C-A)) >= 0) newpoly.push_back(C);
      if(dcmp(Cross(B-A, C-D)) != 0) {
         Point ip = GetLineIntersection(A, B-A, C, D-C);
         if(OnSegment(ip, C, D)) newpoly.push_back(ip);
      }
   }
   return newpoly;
}
```

凸包

```
//凸包
int ConvexHull(Point *p, int n, Point *ch) {
        int m = 0;
        sort(p, p+n);
        for(int i = 0; i < n; i++) {
            while(m > 1 && Cross(ch[m-1] - ch[m-2], p[i] - ch[m-2]) <= 0) m--;    //去掉在边
上的点
            ch[m++] = p[i];
```

```
    }
    int k = m;
    for(int i = n-2; i >= 0; i--) {
        while(m > k && Cross(ch[m-1] - ch[m-2], p[i] - ch[m-2]) <= 0) m--;    //
        ch[m++] = p[i];
    }
    if(m > 1) m--;
    return m;
}
```

稳定凸包

```
//判断是否是稳定凸包
bool IsStableConvexHull(Point *ch, int n) {
    if(n == 0) return 0;
    ch[n] = ch[0];
    ch[n+1] = ch[1];
    for(int i = 0; i < n; i++) {
        if(Cross(ch[i] - ch[(i-1+n)%n], ch[i+1] - ch[i]) != 0 && Cross(ch[i+1] - ch[i] , ch[i+2] -
ch[i]) != 0)
            return 0;
    }
    return 1;
}
```

旋转卡壳

```
//旋转卡壳求凸包直径
double RotateCalipers(Point *ch, int n) {
    double ans = - inf;
    ch[n] = ch[0];
    int q = 1;
    for(int i = 0; i < n; i++) {
        while(Cross(ch[i+1] - ch[i], ch[q] - ch[i]) < Cross(ch[i+1] - ch[i], ch[q+1] -ch[i])) q =
(q+1)%n;
        ans = max(ans, max(Length(ch[i] - ch[q]), Length(ch[i+1] -ch[q+1])));
    }
    return ans;
}
```

```
//两凸包的最短距离
double RotateCalipers(Point *p1, int n, Point *p2, int m) {
    int minp1 = 0, maxp2 = 0;
    for(int i = 0; i < n; i++){
        if(p1[i].y < p1[minp1].y) minp1 = i;
    }
    for(int i = 0; i < m; i++){
        if(p2[i].y > p2[maxp2].y) maxp2 = i;
    }
    p1[n] = p1[0];
    p2[m] = p2[0];
    double dis = inf, temp;
    for(int i = 0; i < n; i++) {
        while(dcmp(Cross(p1[minp1] - p1[minp1+1], p2[maxp2+1] - p1[minp1+1]) - Cross(p1[minp1] - p1[minp1+1], p2[maxp2] - p1[minp1+1])) < 0) maxp2 = (maxp2+1) % m;
        dis = min(dis, SegmentDistance(p1[minp1], p1[minp1+1], p2[maxp2], p2[maxp2+1]));
        minp1 = (minp1+1) % n;
    }
    return dis;
}
```

半平面交

```
//半平面交
struct Line{
    Point p;
    Vector v;
    double rad;
    Line () {}
    Line (Point p, Vector v) : p(p), v(v) {
        rad = atan2(v.y,v.x);
    }
    bool operator < (const Line &L) const {
        return rad < L.rad;
    }
};
bool OnLeft(Line L, Point p) {
    return Cross(L.v, p - L.p) > 0;
}
Point GetLineIntersection (Line a, Line b) {
```

```
    Vector u = a.p - b.p;
    double t = Cross(b.v, u) / Cross(a.v, b.v);
    return a.p + a.v*t;
}

int HalfplaneIntersection(Line *L, int n,Point *poly) {
    sort(L, L+n);
    int first,last;
    Point *p = new Point[n];
    Line *q = new Line[n];   //双端队列
    q[first = last = 0] = L[0];
    for(int i = 1; i < n; i++) {
        while(first < last && !OnLeft(L[i], p[last-1])) last--;     //去尾
        while(first < last && !OnLeft(L[i], p[first])) first++;
        q[++last] = L[i];
        if(dcmp(Cross(q[last].v, q[last-1].v)) == 0) {
            last--;
            if(OnLeft(q[last], L[i].p)) q[last] = L[i];
        }
        if(first < last) p[last-1] = GetLineIntersection (q[last-1],q[last]);
    }
    while(first < last && !OnLeft(q[first], p[last-1])) last--;   //删除无用平面
    if(last - first <= 1) {
        delete []p;
        delete []q;
       return 0;   //空集
    }
    p[last] = GetLineIntersection (q[last], q[first]);
    int m = 0;
    for(int i = first; i <= last; i++) poly[m++] = p[i];
    delete []p;
    delete []q;
    return m;
}
```

圆与多边形的面积交

```
//线段 AB 与圆的交点
void LineIntersectionCircle(Point A, Point B, Circle cr, Point *p, int &num){
    double x0 = cr.o.x, y0 = cr.o.y;
    double x1 = A.x, y1 = A.y, x2 = B.x, y2 = B.y;
    double dx = x2 - x1, dy = y2 - y1;
    double a = dx * dx + dy * dy;
```

```
    double b = 2 * dx *(x1 - x0) + 2 * dy * (y1 -y0);
    double c = (x1 - x0) * (x1 - x0) + (y1 - y0) * (y1 - y0) - cr.r *cr.r;
    double delta = b*b - 4*a*c;
    num = 0;
    if(dcmp(delta) >= 0){
        double t1 = (-b - sqrt(delta)) / (2*a);
        double t2 = (-b + sqrt(delta)) / (2*a);
        if(dcmp(t1-1) <= 0 && dcmp(t1) >= 0) {
            p[num++] = Point(x1 + t1*dx, y1 + t1*dy);
        }
        if(dcmp(t2-1) <= 0 && dcmp(t2) >= 0) {
            p[num++] = Point(x1 + t2*dx, y1 + t2*dy);
        }
    }
}
//扇形面积
double SectorArea(Point a, Point b, Circle cr){
    double theta = Angle(a - cr.o) - Angle(b - cr.o);
    while(theta <= 0) theta += 2*pi;
    while(theta > 2*pi) theta -= 2*pi;
    theta = min(theta, 2*pi - theta);
    return cr.r * cr.r * theta / 2;
}

double cal(Point a, Point b, Circle cr){
    Point tp[3];
    int num = 0;
    Vector ta = a - cr.o;
    Vector tb = b - cr.o;
    bool ina = (Length(ta) - cr.r) < 0;
    bool inb = (Length(tb) - cr.r) < 0;
    if(ina){
        if(inb){
            return fabs(Cross(ta, tb))/2;
        } else{
            LineIntersectionCircle(a, b, cr, tp, num);
            return SectorArea(b, tp[0], cr) + fabs(Cross(ta, tp[0] - cr.o))/2;
        }
    } else{
        if(inb){
            LineIntersectionCircle(a, b, cr, tp, num);
            return SectorArea(a, tp[0], cr) + fabs(Cross(tb, tp[0] - cr.o))/2;
        } else {
            LineIntersectionCircle(a, b, cr, tp, num);
```

```
                if(num == 2) {
                    return SectorArea(a, tp[0], cr) + SectorArea(b, tp[1], cr) + fabs(Cross(tp[0]-cr.o, tp[1]-cr.o))/2;
                } else {
                    return SectorArea(a, b, cr);
                }
            }
        }
}
//圆与多边形的面积交
double CirclePolyArea(Point *p, int n, Circle cr){
    double res = 0;
    p[n] = p[0];
    for(int i = 0; i < n; i++){
        int sgn = dcmp(Cross(p[i] - cr.o, p[i+1] - cr.o));
        if(sgn){
            res += sgn * cal(p[i], p[i+1], cr);
        }
    }
    return res;
}
```

圆的面积交

```
double area[maxn];
#define sqr(x) (x)*(x)
int dcmp(double x) {
    if (x < -eps) return -1; else return x > eps;
}
struct cp {
    double x, y, r, angle;
    int d;
    cp(){}
    cp(double xx, double yy, double ang = 0, int t = 0) {
        x = xx;   y = yy;   angle = ang;   d = t;
    }
    void get() {
        scanf("%lf%lf%lf", &x, &y, &r);
        d = 1;
    }
}cir[maxn], tp[maxn * 2];
double dis(cp a, cp b) {
```

```
    return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
}
double cross(cp p0, cp p1, cp p2) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p1.y - p0.y) * (p2.x - p0.x);
}
int CirCrossCir(cp p1, double r1, cp p2, double r2, cp &cp1, cp &cp2) {
    double mx = p2.x - p1.x, sx = p2.x + p1.x, mx2 = mx * mx;
    double my = p2.y - p1.y, sy = p2.y + p1.y, my2 = my * my;
    double sq = mx2 + my2, d = -(sq - sqr(r1 - r2)) * (sq - sqr(r1 + r2));
    if (d + eps < 0) return 0; if (d < eps) d = 0; else d = sqrt(d);
    double x = mx * ((r1 + r2) * (r1 - r2) + mx * sx) + sx * my2;
    double y = my * ((r1 + r2) * (r1 - r2) + my * sy) + sy * mx2;
    double dx = mx * d, dy = my * d; sq *= 2;
    cp1.x = (x - dy) / sq; cp1.y = (y + dx) / sq;
    cp2.x = (x + dy) / sq; cp2.y = (y - dx) / sq;
    if (d > eps) return 2; else return 1;
}
bool circmp(const cp& u, const cp& v) {
    return dcmp(u.r - v.r) < 0;
}
bool cmp(const cp& u, const cp& v) {
    if (dcmp(u.angle - v.angle)) return u.angle < v.angle;
    return u.d > v.d;
}
double calc(cp cir, cp cp1, cp cp2) {
    double ans = (cp2.angle - cp1.angle) * sqr(cir.r)
        - cross(cir, cp1, cp2) + cross(cp(0, 0), cp1, cp2);
    return ans / 2;
}
void CirUnion(cp cir[], int n) {
    cp cp1, cp2;
    sort(cir, cir + n, circmp);
    for (int i = 0; i < n; ++i)
        for (int j = i + 1; j < n; ++j)
            if (dcmp(dis(cir[i], cir[j]) + cir[i].r - cir[j].r) <= 0)
                cir[i].d++;
    for (int i = 0; i < n; ++i) {
        int tn = 0, cnt = 0;
        for (int j = 0; j < n; ++j) {
            if (i == j) continue;
            if (CirCrossCir(cir[i], cir[i].r, cir[j], cir[j].r,
                cp2, cp1) < 2) continue;
            cp1.angle = atan2(cp1.y - cir[i].y, cp1.x - cir[i].x);
            cp2.angle = atan2(cp2.y - cir[i].y, cp2.x - cir[i].x);
```

```
                cp1.d = 1;      tp[tn++] = cp1;
                cp2.d = -1;     tp[tn++] = cp2;
                if (dcmp(cp1.angle - cp2.angle) > 0) cnt++;
            }
            tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, pi, -cnt);
            tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, -pi, cnt);
            sort(tp, tp + tn, cmp);
            int p, s = cir[i].d + tp[0].d;
            for (int j = 1; j < tn; ++j) {
                p = s;   s += tp[j].d;
                area[p] += calc(cir[i], tp[j - 1], tp[j]);
            }
        }
}
```

自适应辛普森积分

```
//eps 一般取 1e-5 就可以了
double F(double x) {
    //Simpson 公式用到的函数
}
double simpson(double a, double b) {   //三点 Simpson 法，这里要求 F 是一个全局函数
    double c = a + (b - a) / 2;
    return (F(a) + 4 * F(c) + F(b))*(b - a) / 6;
}
double asr(double a, double b, double eps, double A) { //自适应 Simpson 公式（递归过程）。已知整个区间[a,b]上的三点 Simpson 值 A
    double c = a + (b - a) / 2;
    double L = simpson(a, c), R = simpson(c, b);
    if (fabs(L + R - A) <= 15 * eps)return L + R + (L + R - A) / 15.0;
    return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
}
double asr(double a, double b, double eps) { //自适应 Simpson 公式（主过程）
    return asr(a, b, eps, simpson(a, b));
}
```

PSGL

```
struct Edge{
    int u, v;
    double ang;
```

```
    Edge(int u=0, int v=0, double ang=0): u(u), v(v), ang(ang){}
};
const int maxe = 10000+10;  // 最大边数

// 平面直线图（PSGL）实现
struct PSGL{
    int n, m, face_cnt;
    double x[maxe], y[maxe];
    vector<Edge> edges;
    vector<int> G[maxe];
    int vis[maxe<<1];  // 每条边是否已经访问过
    int left[maxe<<1];  // 左面的编号
    int pre[maxe<<1];  // 相同起点的上一条边（即顺时针旋转碰到的下一条边）的编号

    vector<Poly> faces;
    double area[maxe];

    void init(int n) {
        this->n = n;
        for(int i = 0; i < n; i++) G[i].clear();
        edges.clear();
        faces.clear();
    }
    double getAngle(int u,int v) {
        return atan2(y[v]-y[u], x[v]-x[u]);
    }
    void add(int u, int v) {
        edges.pb(Edge(u,v,getAngle(u,v)));
        edges.pb(Edge(v,u,getAngle(v,u)));
        m = edges.size();
        G[u].pb(m-2);
        G[v].pb(m-1);
    }
    //找出 faces 并计算面积
    void build(){
        for(int  u = 0; u < n; u++) {
            int d = G[u].size();
            for(int i = 0; i < d; i++) {
                for(int j = i+1; j < d; j++) {
                    if(edges[G[u][i]].ang > edges[G[u][j]].ang) swap(G[u][i],G[u][j]);
                }
            }
            for(int i = 0; i < d; i++) pre[G[u][(i+1)%d]] = G[u][i];
        }
```

```
        memset(vis, 0, sizeof(vis));
        face_cnt = 0;
        for(int u = 0; u < n; u++) {
            for(int i = 0; i < G[u].size(); i++){
                int e = G[u][i];
                if(!vis[e]){   //逆时针找圈
                    face_cnt++;
                    Poly p;
                    while(1){
                        vis[e] = 1;
                        left[e] = face_cnt;
                        int from = edges[e].u;
                        p.pb(Point(x[from], y[from]));
                        e = pre[e^1];
                        if(e == G[u][i]) break;
                    }
                    faces.pb(p);
                }
            }
        }
        for(int i = 0; i < faces.size(); i++) {
            area[i] = GetArea(faces[i]);

        }
    }

};

//poly 删除三点共线的情况
Poly simplify(const Poly& p) {
    Poly temp;
    int n = p.size();
    for(int i = 0; i < n; i++) {
        Point a = p[i];
        Point b = p[(i+1)%n];
        Point c = p[(i+2)%n];
        if(dcmp(Cross(a-b, c-b)) != 0) temp.pb(b);
    }
    return temp;
}
```

三维

点&&向量

```
struct Point3{
    double x, y, z;
    Point3(double x=0, double y=0, double z=0) : x(x), y(y), z(z) {}
};
int dcmp(double x) {
    if(fabs(x) < eps) return 0;
    return x < 0 ? -1 : 1;
}
typedef Point3 Vector3;
Vector3 operator + (Vector3 a, Vector3 b) {
    return Vector3(a.x + b.x, a.y + b.y, a.z + b.z);
}
Vector3 operator - (Point3 a, Point3 b) {
    return Vector3(a.x - b.x, a.y - b.y, a.z - b.z);
}
Vector3 operator * (Vector3 a, double p) {
    return Vector3(a.x * 3, a.y * 3, a.z * 3);
}
Vector3 operator / (Vector3 a, double p) {
    return Vector3(a.x / p, a.y / p, a.z / p);
}
double Dot(Vector3 a, Vector3 b) {
    return a.x * b.x + a.y * b.y + a.z * b.z;
}
bool operator == (Point3 a, Point3 b) {
    return a.x==b.x && a.y==b.y && a.z==b.z;
}
```

基础运算

```
double Length(Vector3 a) {
    return sqrt(Dot(a,a));
}
double Angle(Vector3 a, Vector3 b) {
    return acos(Dot(a,b) / Length(a) / Length(b)) ;
}
Vector3 Normal(Vector3 a) {
```

```
    return a / Length(a);
}
Vector3 Cross(Vector3 a, Vector3 b) {
    return Vector3(a.y*b.z - a.z*b.y, a.z*b.x - a.x*b.z, a.x*b.y - a.y*b.x);
}
double Area2(Point3 a, Point3 b, Point3 c) {
    return Length(Cross(b-a, c-a));
}
```

简单常用函数

```
//点 p 到平面 p0-n 的距离,n 是单位向量
double DistanceToPlane(const Point3 & p, const Point3 &p0, const Vector3 &n) {
    return fabs(Dot(p-p0, n));  //如果不取绝对值，得到的是有向距离
}

//点 P 在平面 p0-n 上的投影， n 必须为单位向量
Point3 GetPlaneProjection(const Point3 p, const Point3 &p0, const Vector3 &n) {
    return p - n * Dot(p - p0, n);
}
//直线 p1-p2 到平面 p0-n 的交点，假定交点唯一存在
Point3 LinePlaneIntersection(Point3 p1, Point3 p2, Point3 p0, Vector3 n) {
    Vector3 v = p2 - p1;
    double t = (Dot(n, p0 -p1) / Dot(n, p2 -p1)); //判断分母是否为 0
    return p1 + v * t; //如果是线段，判断 t 是不是在 0 和 1 之间
}

//点 p 在三角形 p0,p1,p2 中
bool PointInTri(Point3 p, Point3 p0, Point3 p1, Point3 p2) {
    double area1 = Area2(p, p0, p1);
    double area2 = Area2(p, p1, p2);
    double area3 = Area2(p, p2, p0);
    return dcmp(area1 + area2 + area3 - Area2(p0, p1, p2)) == 0;
}
//三角形 p0,p1,p2 是否和线段 ab 相交
bool TriSegIntersection(Point3 p0, Point3 p1, Point3 p2, Point3 a, Point3 b, Point3 &p) {
    Vector3 n = Cross(p1 - p0, p2 - p0);
    if(dcmp(Dot(n, b - a)) == 0) return false;  //无交点
    double t = Dot(n, p0 - a) / Dot(n, b - a);
    if(dcmp(t) < 0 || dcmp(t-1) > 0) return false;  //交点不在直线上
    p = a + (b - a) * t;
    return PointInTri(p, p0, p1, p2);  //是否在三角形内
```

```
}
//点 p 到直线 ab 的距离
double DistanceToLine(Point3 p, Point3 a, Point3 b) {
	Vector3 v1 = b - a, v2 = p - a;
	return Length(Cross(v1, v2)) / Length(v1);
}
//点 p 到线段 ab 的距离
double DisstanceToSegment(Point3 p, Point3 a, Point3 b) {
	if(a == b) return Length(p - a);
	Vector3 v1 = b - a, v2 = p - a, v3 = p - b;
	if(dcmp(Dot(v1, v2)) < 0) return Length(v2);
	else if(dcmp(Dot(v1, v3)) > 0) return Length(v3);
	else return Length(Cross(v1, v2)) / Length(v1);
}
//四面体 abcd 的体积
double Volume6(Point3 a, Point3 b, Point3 c, Point3 d) {
	return Dot(d-a, Cross(b-a, c-a));
}
```

三维凸包

```
//三维凸包
int vis[200][200];
struct Face{
	int v[3];
	Vector3 normal(Point3 *p) const {
		return Cross(p[v[1]] - p[v[0]], p[v[2]] - p[v[0]]);
	}
	int cansee(Point3 *p, int i) const {
		return Dot(p[i] - p[v[0]], normal(p)) > 0 ? 1 : 0;
	}
};
//倍增法求三维凸包
//没有考虑特殊情况(如四点共面),实践中,请在调用之前对输入点进行微小扰动
vector<Face> CH3D(Point3 *p, int n) {
	vector<Face> cur;
	memset(vis, 0, sizeof(vis));
	//由于已经扰动,前三个点不共线
	cur.push_back((Face){ {0, 1, 2} } );
	cur.push_back((Face){ {2, 1, 0} } );
	for(int i = 3; i < n; i++) {
		vector<Face> next;
```

```
            for(int j = 0; j < cur.size(); j++) {
                Face &f = cur[j];
                int res = f.cansee(p, i);
                if(!res) next.push_back(f);
                for(int k = 0; k < 3; k++) vis[f.v[k]][f.v[(k + 1) % 3]] = res;
            }
            for(int j = 0; j < cur.size(); j++) {
                for(int k = 0; k < 3; k++) {
                    int a = cur[j].v[k], b = cur[j].v[(k+1)%3];
                    if(vis[a][b] != vis[b][a] && vis[a][b])  //(a, b) 是分界线, 左边对(a, b) 可见
                        next.push_back((Face) { {a, b, i} } );
                }
            }
            cur = next;
        }
        return cur;
}
//扰动!!!
double rand01() {
    return rand() / (double)RAND_MAX;
}
double randeps() {
    return (rand01() - 0.5) * eps;
}
Point3 add_noise(Point3 p) {
    return Point3(p.x + randeps(), p.y + randeps(), p.z + randeps());
}
```

多边形重心

```
/*
 *  求多边形重心
 *  INIT: pnt[]已按顺时针(或逆时针)排好序; | CALL: res = bcenter(pnt, n);
 */
Point bcenter(Point pnt[], int n) {
    Point p, s;
    double tp, area = 0, tpx = 0, tpy = 0;
    p.x = pnt[0].x;
    p.y = pnt[0].y;
    for (int i = 1; i <= n; ++i) {
        //  Point:0 ~ n - 1
        s.x = pnt[(i == n) ? 0 : i].x;
        s.y = pnt[(i == n) ? 0 : i].y;
```

```
            tp = (p.x * s.y - s.x * p.y);
            area += tp / 2;
            tpx += (p.x + s.x) * tp;
            tpy += (p.y + s.y) * tp;
            p.x = s.x;
            p.y = s.y;
        }
        s.x = tpx / (6 * area);
        s.y = tpy / (6 * area);
        return s;
}
```

三角形各种心

已知圆锥表面积 S 求最大体积 V

```
    V = S * sqrt(S / (72 * Pi))
```

三角形重点
设三角形的三条边为 a, b, c, 且不妨假设 a <= b <= c.

面积
三角形面积可以根据海伦公式求得：

```
    s = sqrt(p * (p - a) * (p - b) * (p - c));
    p = (a + b + c) / 2;
```

关键点与 A, B, C 三顶点距离之和

费马点

该点到三角形三个顶点的距离之和最小。
有个有趣的结论:
若三角形的三个内角均小于 120 度,那么该点连接三个顶点形成的三个角均为 120 度;若三角形存在一个内角大于 120 度,则该顶点就是费马点。
计算公式如下:
若有一个内角大于 120 度(这里假设为角 C),则距离为 a + b;若三个内角均小于 120 度,则距离为 sqrt((a * a + b * b + c * c + 4 * sqrt(3.0) * s) / 2)。

内心

角平分线的交点。

令 x = (a + b - c) / 2, y = (a - b + c) / 2, z = (-a + b + c) / 2, h = s / p.
计算公式为 sqrt(x * x + h * h) + sqrt(y * y + h * h) + sqrt(z * z + h * h)。

重心

中线的交点。
计算公式如下:
2.0 / 3 * (sqrt((2 * (a * a + b * b) - c * c) / 4)
+ sqrt((2 * (a * a + c * c) - b * b) / 4) + sqrt((2 * (b * b + c * c) - a * a) / 4))。

垂心

垂线的交点。
计算公式如下:
3 * (c / 2 / sqrt(1 - cosC * cosC))。

外心

三点求圆心坐标。

```
Point waixin(Point a, Point b, Point c)
{
    double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1 * a1 + b1 * b1) / 2;
    double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2 * a2 + b2 * b2) / 2;
    double d = a1 * b2 - a2 * b1;
    return Point(a.x + (c1 * b2 - c2 * b1) / d, a.y + (a1 * c2 -a2 * c1) / d);
}
```

圆&&球相关

白书 p264

```
struct Circle{
    Point c;
    double r;
    Circle(Point c, double r) : c(c), r(r) {}
    Point point(double a){
        return Point(c.x + cos(a) * r, c.y + sin(a) * r);
    }
};
```

直线与圆交点

```
// 直线 AB 和圆心为 C，半径为 r 的圆的交点
// 返回交点个数，t1, t2 分别为两个交点在直线方程中的参数，p1 和 p2 为交点本身
int getLineCircleIntersection(Point A, Point B, Point C, double r, double& t1, double& t2){
   // 初始方程：(A.x + t(B.x - A.x) - C.x)^2 + (A.y + t(B.y - A.y) - C.y)^2 = r^2
   // 整理得：(at + b)^2 + (ct + d)^2 = r^2
   double a = B.x - A.x;
   double b = A.x - C.x;
   double c = B.y - A.y;
   double d = A.y - C.y;
   // 展开得：(a^2 + c^2)t^2 + 2(ab + cd)t + b^2 + d^2 - r^2 = 0，即 et^2 + ft + g = 0
   double e = a * a + c * c;
   double f = 2 * (a * b + c * d);
   double g = b * b + d * d - r * r;
   double delta = f * f - 4 * e * g; // 判别式
   if(dcmp(delta) < 0) return 0; // 相离
   if(dcmp(delta) == 0){ // 相切
      t1 = t2 = -f / (2 * e);
      return 1;
   }
   t1 = (-f - sqrt(delta)) / (2 * e);
   t2 = (-f + sqrt(delta)) / (2 * e);
   return 2;
}
```

切线

```
// 过点 p 到圆 C 的切线。v[i]是第 i 条切线的向量。返回切线条数
int getTangents(Point p, Circle C, Vector* v) {
   Vector u = C.c - p;
   double dist = Length(u);
   if(dist < C.r) return 0;
   else if(dcmp(dist - C.r) == 0) { // p 在圆上，只有一条切线
      v[0] = Rotate(u, PI/2);
      return 1;
   } else {
      double ang = asin(C.r / dist);
      v[0] = Rotate(u, -ang);
      v[1] = Rotate(u, +ang);
      return 2;
   }
}
```

两圆公切线

白书 p267

外接圆

```
Circle CircumscribedCircle(Point p1, Point p2, Point p3) {
   double Bx = p2.x-p1.x, By = p2.y-p1.y;
   double Cx = p3.x-p1.x, Cy = p3.y-p1.y;
   double D = 2*(Bx*Cy-By*Cx);
   double cx = (Cy*(Bx*Bx+By*By) - By*(Cx*Cx+Cy*Cy))/D + p1.x;
   double cy = (Bx*(Cx*Cx+Cy*Cy) - Cx*(Bx*Bx+By*By))/D + p1.y;
   Point p = Point(cx, cy);
   return Circle(p, Length(p1-p));
}
```

内切圆

```
Circle InscribedCircle(Point p1, Point p2, Point p3) {
   double a = Length(p2-p3);
   double b = Length(p3-p1);
   double c = Length(p1-p2);
   Point p = (p1*a+p2*b+p3*c)/(a+b+c);
   return Circle(p, DistanceToLine(p, p1, p2));
}
```

球

```
//角度转换成弧度
double torad(double deg){
      return deg / 180 * acos(-1);
}
//经纬度（角度）转换成空间坐标
void get_coord(double R, double lat, double lng, double &x, double &y, double &z){
      lat = torad(lat);
      lng = torad(lng);
      x = R * cos(lat) * cos(lng);
      y = R * cos(lat) * sin(lng);
      z= R * sin(lat);
}
```

图论：

最短路

Dijkstra

优先队列+前向星优化

```
typedef pair<int, int> PII;
void dijkstra(int s, int *dis, int *par) {
	priority_queue<PII, vector<PII>, greater<PII> > pq;
	for(int i = 0; i < maxv; i++) dis[i] = inf;
	dis[s] = 0;
	pq.push(PII(0, s));
	while(!pq.empty()) {
		PII temp = pq.top();
		pq.pop();
		int u = temp.second;
		if(dis[u] < temp.first) continue;
		for(int i = head[u]; i != -1; i = e[i].nex) {
			if(dis[e[i].v] > dis[u]+e[i].w){
				dis[e[i].v] = dis[u]+e[i].w;
				par[e[i].v] = i;
				pq.push(PII(dis[e[i].v], e[i].v));
			}
		}
	}
}
```

Bellman_ford

判负环

```
int inq[maxv], deg[maxv], dis[maxv];
bool bellman_ford() {
	queue<int> q;
	memset(inq, 0, sizeof(inq));
	memset(deg, 0, sizeof(deg));
	for(int i = 0; i < n; i++) {
		dis[i] = 0;
		inq[i] = 1;
```

```
            q.push(i);
            deg[i] = 1;
        }
        while(!q.empty()) {
            int u = q.front();
            q.pop();
            inq[u] = 0;
            for(int i = head[u]; i != -1; i=e[i].nex) {
                int v = e[i].v;
                if(dis[v] > dis[u] + e[i].w) {
                    dis[v] = dis[u] + e[i].w;
                    if(!inq[v]) {
                        inq[v] = 1;
                        q.push(v);
                        if(++deg[v] >= n) return 1;   //有负环 (不知道等号是否成立)
                    }
                }
            }
        }
        return 0;
}
```

生成树

最小树形图（朱刘算法）

```
struct Edge{
    int u, v, w;
    Edge(int u = 0, int v = 0, int w = 0) : u(u), v(v), w(w) {}
}e[maxe];

int   pre[maxn], id[maxn], vis[maxn];
LL in[maxn];
/*
 * n 个点（标号 1 到 n ）(有时要添加虚拟节点 0 )
 *  m条边 ( 0 到m － 1 )
 */
 LL Dir_MST(int n, int m){
    int rt = 0;
    LL ans = 0;
    for(int i = 1; i <= n; i++)   e[m++] = Edge(rt, i, 0);   //添加虚拟节点 rt = 0
```

```
    n++;
    int cnt = m;
    while(true) {
        //找最小入边
        for(int i = 0; i < n; i++) in[i] = inf;
        for(int i = 0; i < cnt; i++) {
            int u = e[i].u, v = e[i].v, w = e[i].w;
            if(w < in[v] && u != v) {
                in[v] = w;
                pre[v] = u;
            }
        }
        for(int i = 0; i < n; i++){
            if(i == rt) continue;
            if(in[i] == inf)       return -1; //  有点没有入边，无法到达
        }
        //找环
        int cntnode = 0;
        memset(vis, -1, sizeof(vis));
        memset(id, -1, sizeof(id));
        in[rt] = 0;
        for(int i = 0; i < n; i++){       //标记每个环
            ans = ans + in[i];
            int v = i;
            while(vis[v] != i && v != rt && id[v] == -1) {   //每个点寻找其前序点，要么最终寻找至根部，要么找到一个环
                vis[v] = i;
                v = pre[v];
            }
            if(id[v] == -1 && v != rt){       //缩点
                for(int u = pre[v]; u != v; u = pre[u]) id[u] = cntnode;
                id[v] = cntnode++;
            }
        }
        if(cntnode == 0) break;       //无环     则 break

        //建立新图
        for(int i = 0; i < n; i++){
            if(id[i] == -1) id[i] = cntnode++;
        }
        for(int i = 0; i < cnt; i++){
            int v = e[i].v;
            e[i].u = id[e[i].u];
            e[i].v = id[e[i].v];
```

```
                if(e[i].u != e[i].v) e[i].w -= in[v];
            }
            n = cntnode;
            rt = id[rt];
        }
        return ans;
}
```

二分图

KM

无 slack

```
int c[maxn][maxn];   //权值
int vg[maxn], vb[maxn]; //vis
int eb[maxn], eg[maxn];   //期望
int mc[maxn];   //匹配结果
int d;

int dfs(int id){
        vg[id]=1;
        for(int i = 0; i < n; i++){
            if(vb[i]) continue;   //只能访问一次
            int gap = eg[id] + eb[i] - c[id][i];
            if(gap == 0){   //有边
                vb[i] = 1;
                if(mc[i] == -1 || dfs(mc[i])){
                    mc[i] = id;
                    return 1;
                }
            }else{
                d = min(d, gap);   //该男最少还需要的期望值
            }
        }
        return 0;
}

void KM(){
        memset(mc, -1, sizeof(mc));
        memset(eb, 0, sizeof(eb));
        for(int i = 0; i < n; i++){
```

```
            eg[i] = c[i][0];
            for(int j = 1; j < n; j++) eg[i] = max(c[i][j], eg[i]);
        }
        for(int i = 0; i < n; i++){
            while(1){
                memset(vg, 0, sizeof(vg));
                memset(vb, 0, sizeof(vb));
                d = inf;
                if(dfs(i)) break;
                for(int i = 0; i < n; i++){
                    if(vg[i]) eg[i] -= d;
                    if(vb[i]) eb[i] += d;
                }
            }
        }
}
```

有 slack

```
int c[maxn][maxn];
int vg[maxn], vb[maxn];
int eb[maxn], eg[maxn];
int mc[maxn];
int slack[maxn];

int dfs(int id){
        vg[id] = 1;
        for(int i = 0; i < n; i++){
            if(vb[i]) continue;   //只能访问一次
            int gap = eg[id] + eb[i] - c[id][i];
            if(gap == 0){   //有边
                vb[i] = 1;
                if(mc[i] == -1 || dfs(mc[i])){
                    mc[i] = id;
                    return 1;
                }
            }else{
                slack[i] = min(slack[i], gap);   //该男最少还需要的期望值
            }
        }
        return 0;
}

void KM(){
        memset(mc, -1, sizeof(mc));
```

```
    memset(eb, 0, sizeof(eb));
    for(int i = 0; i < n; i++){
        eg[i] = c[i][0];
        for(int j = 1; j < n; j++) eg[i] = max(c[i][j], eg[i]);
    }
    for(int i = 0; i < n; i++){
        memset(slack, inf, sizeof(slack));
        while(1){
            memset(vg, 0, sizeof(vg));
            memset(vb, 0, sizeof(vb));
            if(dfs(i)) break;
            int d = inf;
            for(int i = 0; i < n; i++) if(!vb[i]) d = min(d, slack[i]);
            for(int i = 0; i < n; i++){
                if(vg[i]) eg[i] -= d;
                if(vb[i]) eb[i] += d;
                else slack[i] -= d;
            }
        }
    }
}
```

匈牙利算法

```
int vb[maxv], vg[maxv], mcb[maxv], mcg[maxv];
int Hungary(int u) {
    vg[u] = 1;
    for(int i = head[u]; ~i; i = e[i].nex){
        int v = e[i].v;
        if(!vb[v]){
            vb[v] = 1;
            if(mcb[v] == -1 || Hungary(mcb[v])){
                mcb[v] = u;
                mcg[u] = v;
                return 1;
            }
        }
    }
    return 0;
}
```

最小顶点覆盖

二分图的最小顶点覆盖数等于最大匹配数。

且选择的顶点为：

从左边未被匹配的点开始扩展匈牙利树，标记树中的所有节点，取左边未被标记的和右边被标记的。

最大独立集

最大独立集数=总点数-最小顶点覆盖数=总点数-最大匹配数

最小路径覆盖

在图中找尽量少的路径，使得每个节点恰好在一条路径上。换句话说，不同的路径不能有公共点

单独的点也可以作为一条路径。

把每个点都拆成两个点 i 和 i'，如果点 i 可以到点 j，那么连一条边从 i 到 j'

求得该图的最大匹配 m，则最下路径覆盖就是 n-m

最大权闭合子图

（活跃值是活动（左）产生的总活跃值减去邀请学生（右）所花费的活跃值）
源点到正点建正边，负点到汇点建正边，其他点按题意建∞的边即可。
最大权闭合子图的权值等于所有正权点之和减去最小割。

网络流

建边

```
#define CLR(m, a) memset(m, a, sizeof(m))
const int maxv = 610;
const int maxe = 100010;
const int inf = 0x3f3f3f3f;
```

```
struct Edge {
    int u, v, flow, cap;
    int nex;
    Edge(int u=0, int v=0, int flow=0, int cap=0, int nex=0):
        u(u), v(v), flow(flow), cap(cap), nex(nex){}
}e[maxe<<2];
int head[maxv];
int cnt=0;
void init(){
    CLR(head, -1);
    cnt = 0;
}
void add(int u, int v, int cap) {
    e[cnt] = Edge(u, v, 0, cap, head[u]);
    head[u] = cnt++;
    e[cnt] = Edge(v, u, 0, 0, head[v]);
    head[v] = cnt++;
}
```

Dinic

```
int vis[maxv],d[maxv],cur[maxv];
int s,t;   //这里设成了全局变量，用此模板前需要初始化源和汇

int bfs() {
    CLR(vis, 0);
    queue<int> q;
    q.push(s);
    d[s] = 0;
    vis[s] = 1;
    while(!q.empty()) {
        int u = q.front();
        q.pop();
        for(int i = head[u]; ~i; i = e[i].nex) {
            int v = e[i].v;
            if(!vis[v] && e[i].cap > e[i].flow) {
                vis[v]=1;
                d[v] = d[u] + 1;
                q.push(v);
            }
        }
    }
    return vis[t];
```

```
}

int dfs(int x, int a) {
    if(x==t || a==0) return a;
    int flow=0, f;
    for(int &i = cur[x]; ~i; i = e[i].nex) {
        int v = e[i].v;
        if(d[x]+1 == d[v] && (f=dfs(v,min(a,e[i].cap-e[i].flow))) > 0) {
            e[i].flow += f;
            e[i^1].flow -= f;
            flow += f;
            a -= f;
            if(a == 0) break;
        }
    }
    return flow;
}

int Dinic() {
    int flow = 0;
    while(bfs()) {
        for(int i = 0; i < n; i++) cur[i] = head[i];
        flow += dfs(s, inf);
    }
    return flow;
}
```

ISAP

```
int d[maxv], num[maxv], cur[maxv], p[maxv];
int vis[maxv];
int s, t;   //源和汇
int N;   //总的点数

// 预处理，反向 BFS 构造 d 数组
void bfs(){
    CLR(d, -1);
    CLR(vis, 0);
    queue<int> q;
    q.push(t);
    d[t] = 0;
```

```
    vis[t] = 1;
    while(!q.empty()) {
        int u = q.front();
        q.pop();
        for(int i = head[u]; ~i; i = e[i].nex) {
            int id = i&(-2);
            int v = e[id].u;
            if(!vis[v] && e[id].cap > e[id].flow) {
                vis[v] = 1;
                d[v] = d[u] + 1;
                q.push(v);
            }
        }
    }
    //
}
//增广
int augment(){
    int u = t, a = inf;
    while(u != s) {
        int id = p[u];
        a = min(a, e[id].cap - e[id].flow);
        u = e[id].u;
    }
    u = t;
    while(u != s){
        int id = p[u];
        e[id].flow += a;
        e[id^1].flow -= a;
        u = e[id].u;
    }
    return a;
}

int ISAP(){
    bfs();
    int flow = 0;
    CLR(num, 0);
    for(int i = 0; i < N; i++){
        cur[i] = head[i];
        if(~d[i]) num[d[i]]++;     // !
    }
    int u = s;
    while(d[s] < N){
```

```
        if(u == t){
            flow += augment();
            u = s;
        }
        int ok = 0;
        for(int i = cur[u]; ~i; i = e[i].nex){
            int v = e[i].v;
            if(d[u] == d[v] + 1 && e[i].cap > e[i].flow){
                p[v] = i;
                ok = 1;
                cur[u] = i;
                u = v;
                break;
            }
        }
        if(!ok) {
            int m = N - 1;
            for(int i = head[u]; ~i; i = e[i].nex){
                if(e[i].cap > e[i].flow && ~d[e[i].v]) m =min(m, d[e[i].v]);   // ! !
            }
            if(--num[d[u]] == 0) break;   //gap 优化
            num[d[u] = m+1]++;
            cur[u] = head[u];
            if(u != s) u = e[p[u]].u;
        }
    }
    return flow;
}
```

费用流（待补）

连通分量

边双连通分量

```
//每个边双连通分量内部无桥
const int maxv=1010;
struct Edge{
```

```
    int v, nex;
    bool iscut;
}e[maxv<<2];
int head[maxv];
int cnt;
void init(){
    memset(head, -1, sizeof(head));
    cnt=0;
}
void add(int u,int v){
    e[cnt].iscut = 0;
    e[cnt].v = v;
    e[cnt].nex = head[u];
    head[u] = cnt++;
}

int pre[maxv];//   时间戳
int bccno[maxv];// 标记该点属于的 bcc
int dfsk, bcc_cnt;
bool vis[maxv];

int dfs(int u, int id){
    int lowu = pre[u] = ++dfsk;
    for(int i = head[u]; ~i; i = e[i].nex){
        int v = e[i].v;
        if(i == (id ^ 1)) continue;//// 反向边
        if(!pre[v]){
            int lowv = dfs(v, i);
            lowu = min(lowv, lowu);
            if(lowv > pre[u]) e[i].iscut = e[i ^ 1].iscut = true;
        }
        else   lowu = min(lowu, pre[v]);
    }
    return lowu;
}
void dfs1(int u){
    vis[u] = 1;
    bccno[u] = bcc_cnt;
    for(int i = head[u]; ~i; i = e[i].nex){
        if(e[i].iscut) continue;
        if(!vis[e[i].v]) dfs1(e[i].v);
    }
}
//n 个点，bcc_cnt 个边双连通分量（从 1 开始）
```

```
//可以有重边
void find_bcc(int n){
	memset(pre, 0, sizeof(pre));
	memset(bccno, 0, sizeof(bccno));
	memset(vis, 0, sizeof(vis));
	dfsk = bcc_cnt = 0;
	for(int i = 0; i < n; i++) if(!pre[i]) dfs(i, -1);
	for(int i = 0; i < n; i++)
		if(!vis[i]) bcc_cnt++, dfs1(i);
}
```

点双连通分量

```
//注意求的是极大子图

//每个点双连通分量内部无割顶

const int maxv=1010;
const int maxe=1000010;
int n,m;
struct Edge{
	int u, v, nex;
}e[maxe<<1];
int head[maxv];
int cnt;
void add(int u,int v){
	e[cnt].u = u;
	e[cnt].v = v;
	e[cnt].nex = head[u];
	head[u] = cnt++;
}
void init(){
	memset(head, -1, sizeof(head));
	cnt = 0;
}
int pre[maxv];    //时间戳
int iscut[maxv]; //标记是否是割顶
int bccno[maxv]; //标记该点属于哪个连通分量
int dfsk, bcc_cnt;
vector<int> bcc[maxv];    //存连通分量里的点
stack<int> s;    //存边的标号

int dfs(int u, int f){
```

```
    int lowu = pre[u] = ++dfsk;
    int child = 0;
    for(int i = head[u]; ~i; i = e[i].nex){
        int v = e[i].v;
        if(!pre[v]){
            s.push(i);
            child++;
            int lowv = dfs(v, u);
            lowu = min(lowu, lowv);
            if(lowv >= pre[u]){
                iscut[u] = 1;
                bcc_cnt++;
                bcc[bcc_cnt].clear(); //bcc 从 1 开始编号
                for(; ;){
                    int te = s.top();
                    s.pop();
                    Edge p = e[te];
                    if(bccno[p.u] != bcc_cnt) {bcc[bcc_cnt].push_back(p.u); bccno[p.u] =
bcc_cnt;}
                    if(bccno[p.v] != bcc_cnt) {bcc[bcc_cnt].push_back(p.v); bccno[p.v] =
bcc_cnt;}
                    if(p.u==u && p.v==v) break;
                }
            }
        }else if(pre[v] < pre[u] && v != f){
            s.push(i);
            lowu = min(lowu, pre[v]);
        }
    }
    if(f < 0 && child == 1) iscut[u] = 0;
    return lowu;
}

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

强连通分量

```
const int maxv=10010;
const int maxe=100010;
int n,m;
struct Edge{
	int v,nex;
}e[maxe<<1];
int head[maxv];
int cnt;
void init(){
	memset(head, -1, sizeof(head));
	cnt=0;
}
void add(int u, int v){
	e[cnt].v = v;
	e[cnt].nex = head[u];
	head[u] = cnt++;
}
int pre[maxv]; //时间戳
int low[maxv];
int sccno[maxv];   //标记属于哪个强连通分量
int dfsk, scc_cnt;   //scc_cnt 是强连通分量个数，从 1 到 scc_cnt
stack<int> s;

void dfs(int u){
	pre[u] = low[u] = ++dfsk;
	s.push(u);
	for(int i = head[u]; ~i; i = e[i].nex){
		int v = e[i].v;
		if(!pre[v]){
			dfs(v);
			low[u] = min(low[u], low[v]);
		}else if(!sccno[v]) //还在栈里！
			low[u] = min(low[u], low[v]);
	}
	if(pre[u] == low[u]){
		scc_cnt++;
		for(; ;){
			int x = s.top();
			s.pop();
			sccno[x] = scc_cnt;
			if(x == u) break;
```

```
            }
        }
}
void find_scc(int n){
        memset(pre, 0, sizeof(pre));
        memset(low, 0, sizeof(low));
        memset(sccno, 0, sizeof(sccno));
        dfsk = scc_cnt = 0;
        for(int i = 0; i < n; i++) if(!pre[i]) dfs(i);
}
```

TWOSAT

2 - SAT 就是 2 判定性问题，是一种特殊的逻辑判定问题。

选择的置为 1，未选的置为 0

对于 2SAT，每组矛盾都会有四种情况（2*2），题目会限制一种不成立，我们要做的就是找出这一种，用逻辑连接词表示出来，然后取反，加边即可。

```
struct TwoSAT{
        int n;
        vector<int> G[maxn<<1];
        bool mark[maxn<<1];
        int s[maxn<<1], c;

        bool dfs(int x){
                if(mark[x ^ 1]) return false;
                if(mark[x]) return true;
                mark[x] = true;
                s[c++] = x;
                for(int i = 0; i < G[x].size(); i++)
                        if(!dfs(G[x][i])) return false;
                return true;
        }

        void init(int n){
                this->n = n;
                for(int i = 0; i < n * 2; i++) G[i].clear();
                memset(mark, 0, sizeof(mark));
        }

        void add_clause(int x,int xv,int y,int yv){
                x = x * 2 + xv;
```

```
            y = y * 2 + yv;
            G[x ^ 1].push_back(y);
            G[y ^ 1].push_back(x);
        }

        bool solve(){
            for(int i = 0; i < n * 2; i += 2)
                if(!mark[i] && !mark[i ^ 1]){
                    c = 0;
                    if(!dfs(i)){
                        while(c > 0) mark[s[--c]]=false;
                        if(!dfs(i+1)) return false;
                    }
                }
            return true;
        }
}solver;
```

数论

扩展欧几里得

```
// ax+by=gcd(a,b)
// d 为 a 和 b 的最大公因数
void exgcd(LL a, LL b, LL &d, LL &x, LL &y) {
    if(!b) {
        d = a; x = 1; y = 0;
    } else {
        exgcd(b, a%b, d, y, x); y -= x * (a / b);
    }
}
```

如果得到 ax ≡ c (mod b)的某一特解 X，那么令 r = b/gcd(a, b)，可知 x 在[0, r-1]上有唯一解

所以用 x = (X % r + r) % r 就可以求出最小非负整数解 x

一元模线型方程

```
//   求解 n 个方程 x%m[i]=b[i]，返回 x
```

```
int solve(int* b, int* m, int n){
    int m1 = m[0], b1 = b[0];
    for(int i = 1; i < n; i++){
        int m2 = m[i], b2 = b[i];
        int g, x, y;
        exgcd(m1, m2, g, x, y);
        int c = b2 - b1;
        if(c % g) return -1;   //无解
        x = x * (c / g);
        int r = m2 / g;
        x = (x % r + r) % r;
        b1 += m1 * x;
        m1 *= m2 / g;
        b1 %= m1;
    }
    return (b1 % m1 + m1 - 1) % m1 + 1;   //返回正整数解
}
```

中国剩余定理

```
//   求解 x%m[i]=b[i]，其中 m[i]两两互质
int china(int* b, int* m, int n){
    int M = 1, ans = 0;
    for(int i = 0; i < n; i++) M *= m[i];
    for(int i = 0; i < n; i++){
        int N = M / m[i];
        int g, x, y;
        exgcd(N, m[i], g, x, y);
        ans=(ans + b[i] * N * x) % M;
    }
    return (ans + M) % M;
}
```

BSGS

求解 $A^x \equiv B \pmod{C}$，C 是素数

```
#define LL long long
const int mod=99991;
LL head[mod],nex[mod];
LL cnt;
```

```
LL hs[mod],id[mod];
void init(){
    memset(head,-1,sizeof(head));
    cnt=0;
}
void insert_(LL x,LL i){
    hs[cnt]=x;id[cnt]=i;
    LL u=x%mod;
    nex[cnt]=head[u];
    head[u]=cnt++;
}
LL find_(LL x){
    int u=x%mod;
    for(int i=head[u];~i;i=nex[i]){
        if(hs[i]==x) return id[i];
    }
    return -1;
}
//求解 A^x≡B(mod C),返回 x，若无解返回-1
LL bsgs(LL a,LL b,LL c){
    if(b==1) return 0;
    LL m=ceil(sqrt(c*1.0));
    LL q=1;
    for(LL i=0;i<=m;i++){
        if(i==0) insert_(b%c,i);
        else{
            q=q*a%c;
            insert_(q*b%c,i);
        }
    }
    LL temp=1;
    for(LL i=1;i<=m;i++){
        temp=temp*q%c;
        LL j=find_(temp);
        if(j!=-1) return i*m-j;
    }
    return -1;
}
```

扩展 BSGS

求解 $A^x \equiv B \pmod C$

```
#define LL long long
const int mod = 99991;
LL head[mod], nex[mod];
LL cnt;
LL hs[mod], id[mod];
void init() {
	memset(head, -1, sizeof(head));
	cnt = 0;
}
void insert_(LL x, LL i) {
	hs[cnt] = x;
	id[cnt] = i;
	LL u = x % mod;
	nex[cnt] = head[u];
	head[u] = cnt++;
}
LL find_(LL x) {
	int u = x % mod;
	for(int i = head[u]; ~i; i = nex[i]){
		if(hs[i] == x) return id[i];
	}
	return -1;
}
LL gcd(LL a, LL b){
	LL r = a % b;
	while(r){a = b; b = r; r = a % b;}
	return b;
}
//求解 a^x≡b(mod c)，返回 x，无解返回-1
LL BSGS(LL a, LL b, LL c){
	a %= c;
	b %= c;
	if(b == 1) return 0;
	LL g=1, d=1, k=0;
	while((g=gcd(a,c)) != 1){
		if(b % g) return -1;
		b /= g; c /= g; k++;
		d = d * (a / g) % c;
		if(b == d) return k;
	}
	LL m = ceil(sqrt(c*1.0)), q = 1;
	for(LL i = 0; i <= m; i++){
		if(i == 0) insert_(b, i);
```

```
            else {
                q = q * a % c;
                insert_(q * b % c, i);
            }
        }
        for(LL i = 1; i <= m; i++){
            d = d * q % c;
            LL j = find_(d);
            if(j != -1) return i * m - j + k;
        }
        return -1;
}
```

欧拉函数

```
//返回 n 的欧拉函数值
int phi(int n){
        int m = sqrt(n+0.5);
        int ans = n;
        for(int i = 2; i <= m; i++) if(n % i == 0) {
            ans = ans / i * (i - 1);
            while(n % i == 0) n /= i;
        }
        if(n > 1) ans = ans / n * (n-1);
        return    ans;
}

//生成 1 到 n 的欧拉函数值
void get_phi(int n, int *phi) {
        for(int i = 2; i <= n; i++) phi[i] = 0;
        phi[1] = 1;
        for(int i = 2; i <= n; i++) if(!phi[i]){
            for(int j = i; j <= n; j += i){
                if(!phi[j]) phi[j] = j;
                phi[j] = phi[j] / i * (i - 1);
            }
        }
}
```

高斯消元

浮点数消元（待补）

```
void gauss(){
	for(int r = 0; r < tot; r++){
		int max_r = r;
		for(int i = r + 1; i < tot; i++)
			if(fabs(a[i][r]) > fabs(a[max_r][r])) max_r = i;
		if(max_r != r){
			for(int j = 0; j <= tot; j++) swap(a[r][j], a[max_r][j]);
		}
		if(fabs(a[r][r]) < eps) continue;
		//消元
		for(int i = r + 1; i < tot; i++){
			double ta = a[i][r] / a[r][r];
			for(int j = r; j <= tot; j++) a[i][j] -= ta * a[r][j];
		}
		/*
		精度更高
		for(int j = tot; j >= r; j--){
			for(int i = r + 1; i < tot; i++) a[i][j] -= a[i][r] / a[r][r] * a[r][j];
		}
		*/
	}
	//回代
	for(int i = tot - 1; i >= 0; i--){
		double temp = a[i][tot];
		for(int j = i + 1; j < tot; j++)
			temp -= a[i][j] * x[j];
		x[i] = temp / a[i][i];
	}
}
```

模线性方程组

```
const int maxn=75;
const int mod=131;   //模数
int a[maxn][maxn];
int x[maxn];
```

```
int gcd(int a, int b){
    return b == 0 ? a : gcd(b, a % b);
}
int lcm(int a, int b){
    return a / gcd(a, b) * b;
}
void exgcd(int a, int b, int &d, int &x, int &y) {
    if(!b) {
        d = a; x = 1; y = 0;
    } else {
        exgcd(b, a%b, d, y, x); y -= x * (a / b);
    }
}

int gauss(int n, int m){
    int r, c;
    for(r = 0, c = 0; r < n && c < m; c++){
        int max_r = r;
        for(int i = r + 1; i < n; i++) if(abs(a[i][c]) > abs(a[max_r][c])) max_r = i;
        if(max_r != r) for(int j = c; j <= m; j++) swap(a[r][j], a[max_r][j]);
        if(!a[r][c]) continue;
        for(int i = r + 1; i < n; i++) if(a[i][c]){
            int d = lcm(abs(a[i][c]), abs(a[r][c]));
            int t1 = d / a[i][c], t2 = d / a[r][c];
          //  if(a[i][c]*a[r][c]<0) t2=-t2;
            for(int j = c; j <= m; j++) a[i][j] = ((a[i][j] * t1 - a[r][j] * t2) % mod + mod) % mod;
        }
        r++;
    }
    for(int i = r; i < n; i++) if(a[i][m]) return -1; //无解
    for(int i = r - 1; i >= 0; i--){
        x[i] = a[i][m];
        for(int j = i + 1; j < m; j++){
            x[i] = ((x[i] - a[i][j] * x[j]) % mod + mod) % mod;
        }
        int x1, y1, d;
        exgcd(a[i][i], mod, d, x1, y1);
        x1 = ((x1 % mod) + mod) % mod;
        x[i] = x[i] * x1 % mod;
    }
    if(r < m) return m - r;   //自由变元
    return 1;//唯一解
}
```

开关

```
const int maxn=25;
int a[maxn][maxn];
int x[maxn], free_x[maxn];
int n, m;

int gauss(int n, int m){
    int r, c;
    int num = 0;
    for(r = 0, c = 0; r < n && c < m; c++){
        int max_r = r;
        for(int i = r + 1; i < n; i++) if(abs(a[i][c]) > abs(a[max_r][c])) max_r = i;
        if(max_r != r) for(int j = c; j <= m; j++) swap(a[r][j], a[max_r][j]);
        if(!a[r][c]) {free_x[num++] = c; continue;}    //
        for(int i = r + 1; i < n; i++) if(a[i][c]){
            for(int j = c; j <= m; j++) a[i][j] ^= a[r][j];
        }
        r++;
    }
    for(int i = r; i < n; i++) if(a[i][m]) return -1;
    int sta = 1 << (m - r);    //方案数
    int res = inf;
    //枚举自由变元
    for(int i = 0; i < sta; i++){
        int cnt = 0;
        int id = i;
        for(int j = 0; j < m-r; j++){
            x[free_x[j]] = (id&1);
            if(x[free_x[j]]) cnt++;
            id >>= 1;
        }
        for(int j = r - 1; j >= 0; j--){
            int temp = a[j][m];
            for(int k = j + 1; k < m; k++){
                if(a[j][k]) temp ^= x[k];
            }
            x[j] = temp;
            if(x[j]) cnt++;
        }
        if(cnt < res) res = cnt;
    }
    return res;
```

```
}
```

线性基

```
struct LiBase{
    LL a[63], p[63];
    int cnt;   // 重建后，该线性基的范围是[1,1<<(cnt)-1]
    //初始化
    void init(){
        cnt = 0;
        memset(a, 0, sizeof(a));
        memset(p, 0, sizeof(p));
    }
    //插入
    bool insert(LL x){
        for(int i = 62; i >= 0 && x; i--)if(x & (1LL<<i)){
            if(!a[i]){
                a[i] = x;
                break;
            }else x ^= a[i];
        }
        return x > 0;
    }
    //最大
    LL query_max(){
        LL ans = 0;
        for(int i = 62; i >= 0; i--){
            if((ans ^ a[i]) > ans) ans ^= a[i];
        }
        return ans;
    }
    //最小
    LL query_min(){
        for(int i = 0; i <= 62; i++) if(a[i]) return a[i];
        return 0;
    }
    //重建
        void rebuid(){
            for(int i=62;i>=0;i--)
                for(int j=i-1;j>=0;j--) if(a[i]&(1LL<<j)) a[i]^=a[j];
            for(int i=0;i<=62;i++) if(a[i]) p[cnt++]=a[i];
        }
```

```
    //第 k 小
    LL kth(LL k){
        LL ans = 0;
        if(k >= (1LL << cnt)) return -1;
        for(int i = cnt - 1; i >= 0; i--){
            if(k & (1LL << i)) ans ^= p[i];
        }
        return ans;
    }
};
//合并
LiBase merge_(LiBase &x, LiBase &y){
    for(int i = 62; i >= 0; i--)if(y.a[i]) x.insert_(y.a[i]);
    return x;
}
```

01 字典树

```
const int maxnode = 1010 * 32;
const int sigma = 2;
struct Tree01{
    int ch[maxnode][sigma];
    int val[maxnode];
    int num[maxnode];
    int sz;
    void init(){
        sz = 1;
        val[0] = 0;
        num[0] = 0;
        memset(ch[0], 0, sizeof(ch[0]));
    }
    void add(int x){
        int u = 0, n = 31;
        for(int i = n; i >= 0; i--){
            int c = (x >> i) & 1;
            if(!ch[u][c]){
                memset(ch[sz], 0, sizeof(ch[sz]));
                val[sz] = 0;
                num[sz] = 0;
                ch[u][c] = sz++;
            }
            u = ch[u][c];
```

```
            num[u]++;
        }
        val[u] = x;
    }
    //修改（删除或者增加）
    void update(int x, int d){
        int u = 0, n = 31;
        for(int i = n; i >= 0; i--){
            int c = (x >> i) & 1;
            u = ch[u][c];
            num[u] += d;
        }
    }
    //返回与 x 异或最大的值 val[u];
    int query(int x){
        int u = 0, n = 31;
        for(int i = n; i >= 0; i--){
            int c = (x >> i) & 1;
            if(ch[u][c ^ 1] && num[ch[u][c ^ 1]]) u = ch[u][c ^ 1];
            else u = ch[u][c];
        }
        return val[u];
    }
}tre;
```

康托展开

```
//康托展开求法：
//比如 2143 这个数，求其展开：
//从头判断，至尾结束,
//① 比 2（第一位数）小的数有多少个->1 个就是 1，1*3!
//② 比 1（第二位数）小的数有多少个->0 个 0*2!
//③ 比 4（第三位数）小的数有多少个->3 个就是 1,2,3，但是 1,2 之前已经出现，所以是 1*1!
//将所有乘积相加=7
//比该数小的数有 7 个，所以该数排第 8 的位置。
int  fac[] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320}; //i 的阶乘为 fac[i]
//  康托展开-> 表示数字 a 是 a 的全排列中从小到大排，排第几
// n 表示 1~n 个数  a 数组表示数字。
int kangtuo(int n, char a[]){
    int sum = 0;
    for(int i = 0; i < n; i++){
        int t = 0;
```

```
            for(int j = i + 1; j < n; j++)
                if(a[i] > a[j])    t++;
            sum += t * fac[n - i - 1];
        }
        return sum+1;
}
//逆运算的方法：
//假设求 4 位数中第 17 个位置的数字。
//① 17 减去 1   → 16
//② 16  对 3！作除法 → 得 2 余 4
//③   4 对 2！作除法 → 得 2 余 0
//④   0 对 1！作除法 → 得 0 余 0
//据上面的可知：
//我们第一位数（最左面的数），比第一位数小的数有 2 个，显然 第一位数为→ 3
//比第二位数小的数字有 2 个，所以 第二位数为→4
//比第三位数小的数字有 0 个，所以第三位数为→1
//第四位数剩下 2
//该数字为   3412   (正解)
//int    fac[] = {1,1,2,6,24,120,720,5040,40320};
//康托展开的逆运算,{1...n}的全排列，中的第 k 个数为 s[]
void reverse_kangtuo(int n, int k, char s[]){
        int vis[8]={0};
        s[k] = 0;
        --k;
        for(int i = 0; i < n; i++){
            int t = k / fac[n - i - 1];
            int j;
            for (j = 1; j <= n; j++){
                if (!vis[j]){
                    if (t == 0) break;
                    --t;
                }
            }
            s[i] = '0' + j;
            vis[j] = 1;
            k %= fac[n - i - 1];
        }
}
```

n球m盒分配问题

球可分辨	盒子可分辨	盒可空	方案数
是	是	是	m^n
是	是	否	$m!S_2(n,m)$
是	否	是	$\sum_{i=1}^{m} S_2(n,i)$
是	否	否	$S_2(n,m)$
否	是	是	C_{n+m-1}^{m-1}
否	是	否	C_{n-1}^{m-1}
否	否	是	分拆数(n+m,m)
否	否	否	分拆数(n,m)

第一类斯特灵数

把一个包含n个元素的集合分成k个环排列的方法数

初始值$S_1(n,0)=0$，$S_1(1,1)=1$

$$S_1(n+1,k)=S_1(n,k-1)+nS_1(n,k)$$

第二类斯特林数

把一个包含n个元素的集合分成k个非空子集的方法数

初始值$S_2(n,0)=0$，$S_2(n,k)=0(n<k)$，
$S_2(n,1)=S_2(n,n)=1$

$$S_2(n,k)=S_2(n-1,k-1)+kS_2(n-1,k)$$

卡特兰数

一个无穷大的栈的进栈序列为 1,2,3,…,n，问有多少种出栈序列
一个括号序列由 n 个左括号和 n 个右括号组成，问有多少种合法括号序列
把一个正 n 多边形用 n-3 条不相交的对角线划分成 n-2 个三角形的方案数
一棵体积为 n 的有根二叉树有多少种形态
……

递推式

$$h(0)=1$$

$$h(n)=\sum_{i=0}^{n-1}h(i)h(n-1-i)$$

$$h(n)=\frac{4n-2}{n+1}h(n-1)$$

$$h(n)=\frac{C_{2n}^{n}}{n+1}=\frac{(2n)!}{n!(n+1)!}\qquad 0!=1$$

Lucas 定理（待补）

群论（待补）

反素数

对于任何正整数 n，其约数个数记为 f(n)，例如 f(6) = 4，如果某个正整数 n 满足：对任意的正整数 i(0<i<n)，都有 f(i) < f(n)，那么称 n 为反素数。

（1）一个反素数的所有质因子必然是从 2 开始的连续若干个质数，因为反素数是保证约数个数为 x 的这个数 n 尽量小

（2）同理，如果 $n=2^{t_1}\times3^{t_2}\times5^{t_3}\times7^{t_4}\times\ldots$ ，那么必有 $t_1\geq t_2\geq t_3\geq t_4\geq\ldots$

例题：

小明系列故事——未知剩余系
HDU - 4542

题意：两种查询，0 是查询最小正整数 x 满足 1 到 x 中 x 的约数为 k 个，1 是查询最小正整数 x，满足 1 到 x 中 x 的不是 x 的约数的整数有 k 个。

先预处理出 ans[x]，表示不是约数的正整数有 x 个的最小正整数。

```
#include <bits/stdc++.h>
using namespace std;

#define ll long long
const ll inf = (ll)1<<62;
const int maxn=48000;
int pri[]={2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59};
int d[maxn], ans[maxn];
void init(){
    for(int i = 1; i < maxn; i++) {d[i] = i; ans[i] = 0;}
    for(int i = 1; i < maxn; i++){
        for(int j = i; j < maxn; j += i) d[j]--;
        if(!ans[d[i]]) ans[d[i]] = i;
    }
}
ll res;
```

```
int c, k;
void dfs(int d, int ct, ll temp, int num){
    if(num > k || d >= 16) return;
    if(num == k && temp < res) res = temp;
    for(int i = 1; i <= ct; i++){
        if(res / pri[d] < temp || (num * (i + 1) >k)) break;
        temp *= pri[d];
        if(k % (num *(i + 1)) == 0)
            dfs(d + 1, i, temp, num * (i + 1));
    }
}
int main(){
    int t;
    scanf("%d", &t);
    int kase=0;
    init();
    while(t--){
        scanf("%d %d", &c, &k);
        if(c) res = ans[k];
        else {
            res = inf;
            dfs(0, 62, 1, 1);
        }
        printf("Case %d: ",++kase);
        if(res == 0) puts("Illegal");
        else if(res == inf) puts("INF");
        else printf("%lld\n", res);
    }
}
```

字符串

Manacher

```
//返回 s 的最长回文子串的长度
int Manacher(char* s){
    int len = strlen(s);
    for(int i = len; i >= 0; i--){
        s[2 * i + 2] = s[i];
        s[2 * i + 1] = '#';
    }
    s[0] = '*';
```

```
    int id = 0, m = 0;
    for(int i = 2; i < len * 2 + 1; i++){
        if(r[id] + id > i) r[i] = min(r[2 * id - i], r[id] + id - i);
        else r[i] = 1;
        while(s[i - r[i]] == s[i + r[i]]) r[i]++;
        if(id + r[id] < i + r[i]) id = i;
        if(m < r[i]) m = r[i];
    }
    return m-1;
}
```

Trie 树

节点较少时：

```
const int maxnode = 50000*200+10;
const int sigma = 2;
#define CLR(m, a) memset(m, a, sizeof(m))
struct Trie{
    int ch[maxnode][sigma];
    int val[maxnode];
    int sz;
    //添加需要的挂载信息
    void init(){
        CLR(ch[0], 0);
        sz = 1;
    }
    int idx(char c){return c-'0';}
    //插入字符串 s，附加信息为 v（v 必须非零）
    void inser(char *s, int v){
        int u = 0, n = strlen(s);
        for(int i = 0; i < n; i++){
            int c = idx(s[i]);
            if(!ch[u][c]){
                CLR(ch[sz], 0);
                ch[u][c] = sz++;
            }
            u = ch[u][c];
        }
        val[u] = v;
    }
    void find(char* s){
        int u = 0, n = strlen(s);
```

```
        for(int i = 0; i < n; i++){
            int c = idx(s[i]);
            if(!ch[u][c]) return -1;
            u = ch[u][c];
        }
        return val[u];
    }
};
```

左二子右兄弟：

```
const int maxnode = 2000010;
const int sigma = 26;
//Trie 树
struct Trie{
    int head[maxnode],nex[maxnode];
    char ch[maxnode];
    int val[maxnode];
    int sz;

    void init(){
        head[0]=-1;
        sz=1;
    }
    //插入
    void insert(char* s, int v){
        int u = 0, v, n = strlen(s);
        for(int i = 0; i < n; i++){
            int ok = 0;
            for(v = head[u]; ~v; v = nex[v]) if(ch[v] == s[i]){
                ok = 1;
                break;
            }
            if(!ok){
                v=sz++;
                ch[v] = s[i];
                val[v] = 0;
                nex[v] = head[u];
                head[u] = v;
                head[v] = -1;
            }
            u = v;
        }
        val[u] = v;
    }
```

```
    //查询
    int query(char* s){
        int u = 0, v, n = strlen(s);
        for(int i = 0; i < n; i++){
            int ok = 0;
            for(v = head[u]; ~v; v = nex[v]) if(ch[v] == s[i]){
                ok = 1;
                u = v;
                break;
            }
            if(!ok) return -1;
        }
        return val[u];
    }
};
```

Kmp

```
//p 是模板串
const int maxn = 1010;
char s[maxn], p[maxn];
int f[maxn];
int lens, lenp;
void getfail(char* p){
    f[0] = f[1] = 0;
    for(int i = 1; i < lenp; i++){
        int j = f[i];
        while(j && p[j] != p[i]) j = f[j];
        f[i + 1] = (p[i] == p[j] ? j+1 : 0);
    }
    return ;
}
int KMP(char* s, char* p){
    lens = srelen(s);
    lenp = strlen(p);
    getfail(p);
    int j = 0;
    for(int i = 0; i < lens; i++){
        while(j && s[i] != p[j]) j = f[j];
        if(s[i] == p[j]) j++;
        if(j == lenp) return i - m + 1;     //找到 p
    }
```

```
    return -1;   //没找到 p
}
```

扩展 kmp

```
const int maxn = 500010;
char s[maxn], t[maxn];
int exs[maxn], ext[maxn];
int lens, lent;
int nex[maxn];

void getnex(char* t){
    nex[0] = lens;
    int a = 0, p = 0;
    for(int i = 1; i < lens; i++){
        if(i >= p || i + nex[i-a] >= p){
            if(i >= p) p = i;
            while(p < lens && t[p] == t[p-i]) p++;
            nex[i] = p - i;
            a = i;
        }else nex[i] = nex[i - a];
    }
}

ex 为每一个 s 的后缀在 t 中匹配的长度
void exkmp(char* s, char* t, int* ex){
    getnex(t);
    int a=0, p=0;
    for(int i = 0; i < lens; i++){
        if(i >= p || i + nex[i - a] >= p){
            if(i >= p) p = i;
            while(p < lens && p - i < lens && s[p] == t[p-i]) p++;
            ex[i] = p-i;
            a = i;
        }else ex[i] = nex[i - a];
    }
}
```

Ac 自动机

```
const int maxnode = 11000;
```

```
const int sigma = 26;
const int maxn = 160; //

struct AC {
    int ch[maxnode][sigma];
    int f[maxnode], val[maxnode], last[maxnode];
    int sz;
    int cnt[maxn];    //

    void init() {
        CLR(ch[0], 0);
        CLR(cnt, 0);
        sz = 1;
    }
    int idx(char c) {return c-'a';}
    void inser(char *s, int v) {
        int u = 0, n = strlen(s);
        for(int i = 0; i < n; i++){
            int c = idx(s[i]);
            if(!ch[u][c]){
                CLR(ch[sz], 0);
                val[sz] = 0;
                ch[u][c] = sz++;
            }
            u = ch[u][c];
        }
        val[u] = v;
    }
    void print(int u) {
        if(u) {
            cnt[val[u]]++;
            print(last[u]);
        }
    }
    void find(char *s) {
        int n = strlen(s);
        int u = 0;
        for(int i = 0; i < n; i++){
            int c = idx(s[i]);
            while(u && !ch[u][c]) u = f[u];
            u = ch[u][c];
            if(val[u]) print(u);
            else if(last[u]) print(last[u]);
        }
```

```
    }
    void getfail() {
        queue<int> q;
        f[0] = 0;
        for(int c = 0;c < sigma; c++){
            int u = ch[0][c];
            if(u){
                f[u]=0;
                q.push(u);
                last[u] = 0;
            }
        }
        while(!q.empty()){
            int r = q.front();
            q.pop();
            for(int c = 0; c < sigma; c++){
                int u = ch[r][c];
                if(!u) continue;
                q.push(u);
                int v = f[r];
                while(v && !ch[v][c]) v = f[v];
                f[u] = ch[v][c];
                last[u] = val[f[u]] ? f[u] : last[f[u]];
            }
        }
    }
};
```

后缀数组

倍增

注意原串最后一定要补一个 0

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1001;

//sa[i] ：排名第 i 的后缀在哪（i 从 1 开始）
//rk[i]：后缀 i 排第几 （i 从 0 开始）
//h[i]：排名为 i 和 i-1 的两个后缀的最长公共前缀(LCP)长度 （i 从 2 开始）
int s[maxn];
```

```
int sa[maxn], t1[maxn],t2[maxn],c[maxn];
int h[maxn],rk[maxn];
void build_sa(int n, int m){
    int i, *x = t1, *y = t2;
    for(i = 0; i < m; i++) c[i] = 0;
    for(i = 0; i < n; i++) c[x[i] = s[i]]++;
    for(i = 1; i < m; i++) c[i] += c[i-1];
    for(i = n-1; i >= 0; i--) sa[--c[x[i]]] = i;

    for(int k = 1; k <= n; k <<= 1){
        int p = 0;
        for(i = n-k; i < n; i++) y[p++] = i;
        for(i = 0; i < n; i++) if(sa[i] >= k) y[p++] = sa[i]-k;
        for(i = 0; i < m; i++) c[i] = 0;
        for(i = 0; i < n; i++) c[x[y[i]]]++;
        for(i = 1; i< m; i++) c[i] += c[i-1];
        for(i = n-1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];
        swap(x,y);
        p = 1;
        x[sa[0]] = 0;
        for(i = 1; i < n; i++)
           x[sa[i]] = y[sa[i]]==y[sa[i-1]] && y[sa[i]+k]==y[sa[i-1]+k]? p-1 : p++;
        if(p>=n) break;
        m=p;
    }
}

void geth(int n){
    int i, j, k=0;
    for(i = 1; i <= n; i++) rk[sa[i]]=i;
    for(i = 0; i < n; i++) {
        if(k) k--;
        j = sa[rk[i]-1];
        while(s[i+k]==s[j+k]) k++;
        h[rk[i]] = k;
    }
}

int dp[maxn][20];
void rmq_init(int n){
    int k = 0;
    while(1<<(k+1) <= n) k++;
    for(int i = 1; i <= n; i++) dp[i][0] = h[i];
    for(int i = 1; i <= k; i++){
```

```
            for(int j = 1; j + (1<<i) - 1 <= n; j++){
                dp[i][j] = min(dp[i][j-1], dp[i+(1<<j-1)][j-1]);
            }
        }
}
int rmq(int a, int b){
        if(a > b) swap(a, b);    //
        a++;
        int k = 0;
        while(1<<(k+1) <= b-a+1) k++;
        return min(dp[a][k], dp[b - (1<<k) + 1][k]);
}
char str[maxn];
int main(){
        scanf("%s", str);
        int n = strlen(str);
        for(int i = 0; i <    n; i++) s[i] = str[i] - 'a' + 1;
        build_sa(n+1, 300);
        geth(n);
//        for(int i = 1; i <= n; i++)    cout<<sa[i]<<" ";
//        cout<<endl;
//        for(int i = 0; i < n; i++) cout<<rk[i]<<" ";
//        cout<<endl;
//        for(int i = 2; i <= n; i++)    cout<<h[i]<<" ";
//        cout<<endl;
//        int a, b;
        rmq_init(n);
        while(scanf("%d %d", &a, &b))
            cout<<rmq(rk[a], rk[b])<<endl;
}
```

Dc3

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
/*
 * 后缀数组
 * DC3 算法,复杂度 O(n)
 * 所有的相关数组都要开三倍
```

```
 */
const int MAXN = 1000010;
#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 * 3], wb[MAXN * 3], wv[MAXN * 3], wss[MAXN * 3];

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

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

void sort(int *r, int *a, int *b, int n, int m) {
    int i;
    for(i = 0; i < n; i++) wv[i] = r[a[i]];
    for(i = 0; i < m; i++) wss[i] = 0;
    for(i = 0; i < n; i++) wss[wv[i]]++;
    for(i = 1; i < m; i++) wss[i] += wss[i - 1];
    for(i = n - 1; i >= 0; i--) b[--wss[wv[i]]] = a[i];
}

void dc3(int *r, int *sa, int n, int m) {
    int i, j, *rn = r + n;
    int *san = sa + n, ta = 0, tb = (n+1)/3, tbc = 0, p;
    r[n] = r[n+1] = 0;
    for (i = 0; i < n; i++) {
        if (i % 3 != 0) wa[tbc++] = i;
    }
    sort(r + 2, wa, wb, tbc, m);
    sort(r + 1, wb, wa, tbc, m);
    sort(r, wa, wb, tbc, m);
    for(p = 1, rn[F(wb[0])] = 0, i = 1; i < tbc; i++){
        rn[F(wb[i])] = c0(r, 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(n % 3 == 1) wb[ta++] = n - 1;
        sort(r, 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, r, wa[i], wb[j]) ? wa[i++] : wb[j++];
        for (; i < ta; p++) sa[p] = wa[i++];
        for (; j < tbc; p++) sa[p] = wb[j++];
}

//   str 和 sa 也要三倍
void da(int str[], int sa[], int rk[], int h[], int n, int m){
        for(int i = n; i < n * 3; i++) str[i] = 0;
        dc3(str, sa, n+1, m);
        int i, j, k = 0;
        for(i = 0; i <= n; i++) rk[sa[i]] = i;
        for(i = 0; i < n; i++) {
            if(k) k--;
            j = sa[rk[i] - 1];
            while (str[i + k] == str[j + k]) k++;
            h[rk[i]] = k;
        }
}
//sa[i] ：排名第 i 的后缀在哪（i 从 1 开始）
//rk[i]：后缀 i 排第几 （i 从 0 开始）
//h[i]：排名为 i 和 i-1 的两个后缀的最长公共前缀(LCP)长度 （i 从 2 开始）
int s[MAXN * 3];
char str[MAXN * 3];
int sa[MAXN * 3];
int h[MAXN * 3],rk[MAXN * 3];

int main(){
        //freopen("in.txt", "r", stdin);
        while(scanf("%s", str)){
            if(str[0] == '.') break;
            int n = strlen(str);
            for(int i = 0; i <    n; i++) s[i] = str[i] - 'a' + 1;
            da(s, sa, rk, h, n, 300);
            int k;
            for(k = 1; k <= n; k++){
                if(n%k==0 && rk[0] == rk[k]+1 && h[rk[0]] == n-k) break;
            }
            if(k == n+1) k--;
```

```
            printf("%d\n", n/k);
        }
}
```

博弈

BASH 博弈

//巴什博奕 :只有一堆 n 个物品, 两个人轮流从中取物, 规定每次最少取一个, 最多取 m 个, 最后取光者为胜。

```
#include <iostream>
using namespace std;
int main()
{
    int n,m;
    while(cin>>n>>m)
      if(n%(m+1)==0)   cout<<"后手必胜"<<endl;
      else cout<<"先手必胜"<<endl;
    return 0;
}
```

威佐夫博弈（Wythoff Game）：

//有两堆各若干的物品，两人轮流从其中一堆取至少一件物品，至多不限，或从两堆中同时取相同件物品，规定最后取完者胜利。
//直接说结论了，若两堆物品的初始值为（x，y），且 x<y，则另 z=y-x；
//记 w=（int）[（(sqrt（5）+1）/2）*z]；
//若 w=x，则先手必败，否则先手必胜。

尼姆博弈（Nimm Game）：

尼姆博弈指的是这样一个博弈游戏 : 有任意堆物品，每堆物品的个数是任意的，双方轮流从中取物品，每一次只能从一堆物品中取部分或全部物品，最少取一件，取到最后一件物品的人获胜。
结论就是 :把每堆物品数全部异或起来, 如果得到的值为 0, 那么先手必败, 否则先手必胜。

斐波那契博弈：

有一堆物品，两人轮流取物品，先手最少取一个，至多无上限，但不能把物品取完，之后每次取的物品数不能超过上一次取的物品数的二倍且至少为一件，取走最后一件物品的人获胜。
结论是：先手胜当且仅当 n 不是斐波那契数（n 为物品总数）

SG 函数

```
//SG 函数打表
const int MAX_DIG = 64;

//   SG 打表
//   f[]:可以取走的石子个数
//   sg[]:0~n 的 SG 函数值
//   hash[]:mex{}
int f[MAX_DIG];
int sg[MAX_DIG];
int hash[MAX_DIG];

void getSG(int n)
{
      memset(sg, 0, sizeof(sg));
      for (int i = 1; i <= n; i++)
      {
            memset(hash, 0, sizeof(hash));
            for (int j = 1; f[j] <= i; j++)
            {
                  hash[sg[i - f[j]]] = 1;
            }
            for (int j = 0; j <= n; j++)      //   求 mes{}中未出现的最小的非负整数
            {
                  if (hash[j] == 0)
                  {
                        sg[i] = j;
                        break;
                  }
            }
      }
```

```
}

//SG DFS
const int MAX_DIG = 64;

//   DFS
//   注意 S 数组要按从小到大排序 SG 函数要初始化为-1 对于每个集合只需初始化 1 遍
//   n 是集合 s 的大小 S[i]是定义的特殊取法规则的数组
int s[MAX_DIG];
int sg[MAX_DIG * 100];
int n;

int SG_dfs(int x)
{
	if (sg[x] != -1)
	{
		return sg[x];
	}
	bool vis[MAX_DIG];
	memset(vis, 0, sizeof(vis));
	for (int i = 0; i < n; i++)
	{
		if (x >= s[i])
		{
			SG_dfs(x - s[i]);
			vis[sg[x - s[i]]] = 1;
		}
	}
	int e;
	for (int i = 0; ; i++)
	{
		if (!vis[i])
		{
			e = i;
			break;
		}
	}
	return sg[x] = e;
}
```

杂

Bitset

一,定义和初始化

```
bitset<n> b;                                  //b 有 n 位,每位都为 0;
bitset<n> b(u);                               //b 是 unsigned long 型 u 的副本
bitset<n> b(s);                               //b 是 string 对象 s 中含有 n 位字符串的副本
bitset<n> b(s, pos, n);                       //b 是 s 中从 pos 位置开始的 n 个位置的副本
bitset<n> b(s,pos);          //b 从 s 的 pos 位置开始取值到 s 末尾(注取的值从 b 的右端开始)
```

注：①n 定义的位数在初始化时按初始值填充,赋值超出的范围舍去,空余的以零填充.

②bitset 从 string 对象读入位集时按从右到左的顺序.

二,操作

```
b.any();                                      //查找 b 是否存在 1?
b.none();                                     //b 中不存在 1 吗?
b.count();                                    //b 中 1 的个数
b.size();                                     //b 的位数
b[pos];                                       //访问 b 中 pos 处的数值
b.test(pos);                                  //检测 b 中 pos 处是否为 1
b.set();                                      //把 b 中所有位 置为 1
b.set(pos);                                   //把 b 中 pos 位置为 1
b.reset();                                    //把 b 中所有位置为 0
b.reset(pos);                                 //把 b 中 pos 位置为 0
b.flip();                                     //b 中所有二进制位取反
b.flip(pos);                                  //b 中在 pos 处的二进制位取反
b.to_ulong;                                   //返回一个同值得 unsigned long 值
os << b;                                      //把 b 中位集输出
```

数位 dp

等凹回文

```
#include <bits/stdc++.h>
using namespace std;
```

```
#define ll long long
ll dp[20][20][10][2][2][2];
int bit[20];
int num[20];

ll l,r;

//长度，当前哪一位，前一位数字，是否升，是降，是否回文，限制
ll dfs(int len, int cur, int pre, int up, int down, int sta, int lim){
    if(cur < 0) return up && down && sta;
    if(!lim && dp[len][cur][pre][up][down][sta] != -1)
        return dp[len][cur][pre][up][down][sta];
    int maxv = lim ? bit[cur] : 9;
    ll ans = 0;
    for(int i = 0; i <= maxv; i++){
        num[cur] = i;
        if(len == cur){
            if(i == 0) ans += dfs(len - 1, cur - 1, i, up, down, sta, lim && i == maxv);
            else ans += dfs(len, cur - 1, i, up, down, sta, lim && i == maxv);
        }
        else if(i == pre){
            if(sta && cur < (len+1) / 2)
                ans += dfs(len, cur - 1, i, up, down, i == num[len - cur], lim && i == maxv);
            else ans += dfs(len, cur - 1, i, up, down, sta, lim && i == maxv);
        }
        else if(i < pre){
            if(up) continue;
            if(sta && cur < (len+1) / 2)
                ans += dfs(len, cur - 1, i, up, 1, i == num[len - cur], lim && i == maxv);
            else ans += dfs(len, cur - 1, i, up, 1, sta, lim && i == maxv);
        }else if(i > pre){
            if(!down) continue;
            if(sta && cur <(len+1) / 2)
                ans += dfs(len, cur - 1, i, 1, down, i == num[len - cur],lim && i == maxv);
            else ans += dfs(len, cur - 1, i, 1, down, sta, lim && i == maxv);
        }
    }
    if(!lim) dp[len][cur][pre][up][down][sta]=ans;
    return ans;
}

ll solve(ll x){
    int pos=0;
    while(x){
```

```
        bit[pos++] = x % 10;
        x /= 10;
    }
    memset(num, 0, sizeof(num));
    return dfs(pos - 1, pos - 1, 0, 0, 0, 1, 1);
}

int main(){
    int t;
    scanf("%d", &t);
    memset(dp, -1, sizeof(dp));
    while(t--){
        scanf("%lld %lld", &l, &r);
        printf("%lld\n", solve(r) - solve(l-1));
    }
    return 0;
}
```

星期几？

```
/*
 *  已知 1752 年 9 月 3 日是 Sunday，并且日期控制在 1700 年 2 月 28 日后
 */
char name[][15] = { "monday", "tuesday", "wednesday", "thursday", "friday", "saturday",
"sunday"};

int main() {
    int d, m, y, a;
    printf("Day: ");
    scanf("%d", &d);
    printf("Month: ");
    scanf("%d", &m);
    printf("Year: ");
    scanf("%d", &y);
    //  1 月 2 月当作前一年的 13,14 月
    if (m == 1 || m == 2){
        m += 12;
        y--;
    }
    //  判断是否在 1752 年 9 月 3 日之前,实际上合并在一起倒更加省事
    if ((y < 1752) || (y == 1752 && m < 9) || (y == 1752 && m == 9 && d < 3)) {
        //  因为日期控制在 1700 年 2 月 28 日后，所以不用考虑整百年是否是闰年
        a = (d + 2 * m + 3 * (m + 1) / 5 + y + y / 4 + 5) % 7;
```

```
    }
    else {
        //  这里需要考虑整百年是否是闰年的情况
        a = (d + 2 * m + 3 * (m + 1) / 5 + y + y / 4 - y / 100 + y / 400) % 7;  //  实际上这个可以当做公式背下来
    }
    printf("it's a %s\n", name[a]);
    return 0;
}
```

Java

输入输出

```
import java.io.*;
import java.math.*;
import java.util.*;
import java.text.*;
public class Main{
    public static void main(String args[]){
        Scanner cin = new Scanner (new BufferedInputStream(System.in));
        int a;
        double b;
        BigInteger c;
        String s;
        a = cin.nextInt();  //scanf("%d",&a);
        b = cin.nextDouble(); //scanf("%lf",&b);
        c = cin.nextBigInteger();
        s = cin.next();  //scanf("%s",s);
        s = cin.nextLine();  //gets(s);  cin.getline();  读入换行符!
        System.out.println(a+b);  // printf("%lf\n",a+b);
        System.out.print(a+b); // printf("%lf",a+b);
        System.out.printf("%d %.5f\n", a, b);
    }
}
```

大数

```
import java.util.*;
```

```
import java.math.*;

public class Main {
	public static void main(String args[]) {
		int a = 102;
		BigInteger x = BigInteger.valueOf(a);
		Scanner cin = new Scanner(System.in);
		while(cin.hasNext()) {
			BigInteger y = cin.nextBigInteger();
			BigInteger z = x.add(y);
			z = x.multiply(y);
			z = x.subtract(y);
			z = x.divide(y);
			z = x.remainder(y);
			z = x.mod(y);
			z = x.pow(4);
			z = x.gcd(y);
			z = x.abs();
			z = x.negate();
			System.out.println(z.equals(x));
			String s = new String("101010");
			BigInteger u = new BigInteger(s, 2);
			System.out.println(u);
			z = BigInteger.ZERO;
			z = BigInteger.ONE;
			z = BigInteger.TEN;
		}
	}
}
```