Zhejiang University ICPC Team

Routine Library

by WishingBone (Dec. 2002)

Version Information

1.00 Dec2002, initial version by *WishingBone* (Orz)
1.01 Nov2004, updated by *Riveria*
1.02 Oct 2005, updated by *SgHao* & *qherlyt*
1.03 Nov2006, updated by *VegetableB* & *SgHao*
1.04 Oct 2007, updated by *Fire*
1.05 Oct 2008, updated by *watashi*

几何

注意

- 注意舍入方式(0.5 的舍入方向);防止输出-0.
- 几何题注意多测试不对称数据.
- 整数几何注意 xmult 和 dmult 是否会出界;
  符点几何注意 eps 的使用.
- 避免使用斜率;注意除数是否会为 0.
- 公式一定要化简后再代入.
- 判断同一个 2*PI 域内两角度差应该是

abs(a1-a2)<beta||abs(a1-a2)>pi+pi-beta;

相等应该是

abs(a1-a2)<eps||abs(a1-a2)>pi+pi-eps;

- 需要的话尽量使用 atan2,注意:atan2(0,0)=0,
  atan2(1,0)=pi/2,atan2(-1,0)=-pi/2,atan2(0,1)=0,atan2(0,-1)=pi.
- cross product = |u|*|v|*sin(a)
  dot product = |u|*|v|*cos(a)
- (P1-P0)x(P2-P0)结果的意义:

正: <P0,P1>在<P0,P2>顺时针(0,pi)内

负: <P0,P1>在<P0,P2>逆时针(0,pi)内

0 : <P0,P1>,<P0,P2>共线,夹角为 0 或 pi

- 误差限缺省使用 1e-8!

几何公式

- 三角形
  - 半周长 P=(a+b+c)/2
  - 面积 S=aHa/2=absin©/2=sqrt(P(P-a)(P-b)(P-c))
  - 中线 Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2
  - 角平分线 Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)
  - 高线 Ha=bsin©=csin(B)=sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)
  - 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)
  - 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
  - a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2
  - S=D1D2sin(A)/2

  (以下对圆的内接四边形)
  - ac+bd=D1D2
  - 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

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

圆

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

棱柱

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

棱锥

- 体积 V=Ah/3,A 为底面积,h 为高

(以下对正棱锥)

- 侧面积 S=lp/2,l 为斜高,p 为底面周长
- 全面积 T=S+A

棱台

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)

- 侧面积 S=(p1+p2)l/2,p1.p2 为上下底面周长,l 为斜高
- 全面积 T=S+A1+A2

圆柱

- 侧面积 S=2PIrh
- 全面积 T=2PIr(h+r)
- 体积 V=PIr^2h

圆锥

- 母线 l=sqrt(h^2+r^2)
- 侧面积 S=PIrl
- 全面积 T=PIr(l+r)
- 体积 V=PIr^2h/3

圆台

- 母线 l=sqrt(h^2+(r1-r2)^2)
- 侧面积 S=PI(r1+r2)l
- 全面积 T=PIr1(l+r1)+PIr2(l+r2)
- 体积 V=PI(r1^2+r2^2+r1r2)h/3

球

- 全面积 T=4PIr^2
- 体积 V=4PIr^3/3

球台

- 侧面积 S=2PIrh
- 全面积 T=PI(2rh+r1^2+r2^2)
- 体积 V=PIh(3(r1^2+r2^2)+h^2)/6

球扇形

- 全面积 T=PIr(2h+r0),h 为球冠高,r0 为球冠底面半径

- 体积 V=2PIr^2h/3

多边形

```
#include <cstdlib>
#include <cmath>

const double EPS = 1e-8;

struct Point {
    double x, y;
};
struct Line {
    Point a, b;
};

inline bool zero(double x) {
    return ((x > 0 ? x : -x) < EPS);
}

inline int sign(double x) {
    return (x > EPS ? 1 : (x < -EPS ? 2 : 0));
}

inline double xmult(const Point & p1, const Point & p2, const Point & p0) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}

//判定凸多边形，顶点按顺时针或逆时针给出，允许相邻边共线
bool isConvex(int n, const Point * p) {
    int i, s[3] = {1, 1, 1};
    for (i = 0; i < n && s[1] | s[2]; i++) {
        s[sign(xmult(p[(i + 1) % n], p[(i + 2) % n], p[i]))] = 0;
    }
    return (s[1] | s[2]) != 0;
}

//判定凸多边形，顶点按顺时针或逆时针给出，不允许相邻边共线
bool isConvexV2(int n, const Point * p) {
    int i, s[3] = {1, 1, 1};
    for (i = 0; i < n && s[0] && s[1] | s[2]; i++) {
        s[sign(xmult(p[(i + 1) % n], p[(i + 2) % n], p[i]))] = 0;
    }
    return s[0] && s[1] | s[2];
}

//判点在凸多边形内或多边形边上，顶点按顺时针或逆时针给出
//优点：只有加减乘，速度快。缺点：只能处理凸的情况。
```

```
bool insideConvex(const Point & q, int n, const Point * p) {
    int i, s[3] = {1, 1, 1};
    for (i = 0; i < n && s[1] | s[2]; i++) {
        s[sign(xmult(p[(i + 1) % n], q, p[i]))] = 0;
    }
    return (s[1] | s[2]) != 0;
}

//判点在凸多边形内，顶点按顺时针或逆时针给出，在多边形边上返回0
//优点：只有加减乘，速度快。缺点：只能处理凸的情况。
bool insideConvexV2(const Point & q, int n, const Point * p) {
    int i, s[3] = {1, 1, 1};
    for (i = 0; i < n && s[0] && s[1] | s[2]; i++) {
        s[sign(xmult(p[(i + 1) % n], q, p[i]))] = 0;
    }
    return s[0] && s[1] | s[2];
}

//判点在任意多边形内，顶点按顺时针或逆时针给出
//onEdge表示点在多边形边上时的返回值，OFFSET为多边形坐标上限
//优点：大部分情况效率较高。缺点：做任意(random)射线，如rp不好可能要做很多次射线。
const int OFFSET = 10000;

bool insidePolygon(const Point & q, int n, const Point * p, bool onEdge = true) {
    Point q2;
    int i = 0, count = 0;
    while (i < n) {
        q2.x = rand() + OFFSET;
        q2.y = rand() + OFFSET;
        for (count = i = 0; i < n; i++) {
            if (zero(xmult(q, p[i], p[(i + 1) % n])) && (p[i].x - q.x) * (p[(i + 1)
 % n].x - q.x) < EPS && (p[i].y - q.y) * (p[(i + 1) % n].y - q.y) < EPS) {
                return onEdge;
            } else if (zero(xmult(q, q2, p[i]))) {
                break;
            } else if (xmult(q, p[i], q2) * xmult(q, p[(i + 1) % n], q2) < -EPS &&
xmult(p[i], q, p[(i + 1) % n]) * xmult(p[i], q2, p[(i + 1) % n]) < -EPS) {
                count++;
            }
        }
    }
    return count & 1;
}

//判点在任意多边形内，顶点按顺时针或逆时针给出
//onEdge表示点在多边形边上时的返回值
//优点：可以处理所有情况(包括不规则多边形)，缺点：做n次acos，效率可能低一点。
```

```
const double PI = acos(-1.0);

bool insidePolygonV2(const Point & q, int n, const Point * p, bool onEdge = true) {
    double ang = 0.0, x1, y1, x2, y2, dprod, cprod, t;
    for (int i = 0; i < n; ++i) {
        x1 = p[i].x - q.x;
        y1 = p[i].y - q.y;
        x2 = p[(i + 1) % n].x - q.x;
        y2 = p[(i + 1) % n].y - q.y;
        if (fabs(x1) + fabs(y1) < EPS || fabs(x2) + fabs(y2) < EPS) {
            return onEdge;
        }
        dprod = x1 * x2 + y1 * y2;
        cprod = x1 * y2 - x2 * y1;
        t = acos(dprod / (sqrt(x1*x1 + y1*y1) * sqrt(x2*x2 + y2*y2)));
        if (fabs(cprod) < EPS && t > EPS) {
            return onEdge;
        }
        if (cprod < -EPS) {
            t = -t;
        }
        ang += t;
    }
    return fabs(fabs(ang) - PI * 2.0) < EPS;
}

inline bool oppositeSide(const Point & p1, const Point & p2, const Point & l1, cons
t Point & l2) {
    return xmult(l1, p1, l2) * xmult(l1, p2, l2) < -EPS;
}

inline bool dotOnlineIn(const Point & p, const Point & l1, const Point & l2) {
    return zero(xmult(p, l1, l2)) && (l1.x - p.x) * (l2.x - p.x) < EPS && (l1.y - p
.y) * (l2.y - p.y) < EPS;
}

//判线段在任意多边形内，顶点按顺时针或逆时针给出，与边界相交返回 true
const int MAXN = 1000;

bool insidePolygon(const Point & l1, const Point & l2, int n, const Point * p) {
    Point t[MAXN], tt;
    int i, j, k = 0;
    if (!insidePolygon(l1, n, p) || !insidePolygon(l2, n, p)) {
        return false;
    }
    for (i = 0; i < n; i++) {
        if (oppositeSide(l1, l2, p[i], p[(i + 1) % n]) && oppositeSide(p[i], p[(i +
```

```
 1) % n], l1, l2)) {
            return false;
        } else if (dotOnlineIn(l1, p[i], p[(i + 1) % n])) {
            t[k++] = l1;
        } else if (dotOnlineIn(l2, p[i], p[(i + 1) % n])) {
            t[k++] = l2;
        } else if (dotOnlineIn(p[i], l1, l2)) {
            t[k++] = p[i];
        }
    }
    for (i = 0; i < k; i++) {
        for (j = i + 1; j < k; j++) {
            tt.x = (t[i].x + t[j].x) / 2;
            tt.y = (t[i].y + t[j].y) / 2;
            if (!insidePolygon(tt, n, p)) {
                return false;
            }
        }
    }
    return true;
}

Point intersection(const Line & u, const Line & v) {
    Point ret = u.a;
    double t = ((u.a.x - v.a.x) * (v.a.y - v.b.y) - (u.a.y - v.a.y) * (v.a.x - v.b.
x)) / ((u.a.x - u.b.x) * (v.a.y - v.b.y) - (u.a.y - u.b.y) * (v.a.x - v.b.x));
    ret.x += (u.b.x - u.a.x) * t;
    ret.y += (u.b.y - u.a.y) * t;
    return ret;
}

Point barycenter(const Point & a, const Point & b, const Point & c) {
    Line u, v;
    u.a.x = (a.x + b.x) / 2;
    u.a.y = (a.y + b.y) / 2;
    u.b = c;
    v.a.x = (a.x + c.x) / 2;
    v.a.y = (a.y + c.y) / 2;
    v.b = b;
    return intersection(u, v);
}

//多边形重心
Point barycenter(int n, const Point * p) {
    Point ret, t;
    double t1 = 0, t2;
    int i;
```

```
    ret.x = ret.y = 0;
    for (i = 1; i < n-1; i++) {
        if (fabs(t2 = xmult(p[0], p[i], p[i + 1])) > EPS) {
            t = barycenter(p[0], p[i], p[i + 1]);
            ret.x += t.x * t2;
            ret.y += t.y * t2;
            t1 += t2;
        }
    }
    if (fabs(t1) > EPS) {
        ret.x /= t1;
        ret.y /= t1;
    }
    return ret;
}

// 费马点

double fermentpoint(int m, Point p[]) {
    Point u, v;
    double step = 0;
    u.x = u.y = 0;
    for (int i = 0; i < m; ++i) {
        u.x += p[i].x;
        u.y += p[i].y;
    }
    step = u.x + u.y;
    u.x /= m;
    u.y /= m;
    double nowbest = 0;;
    for (int i = 0; i < m; ++i) {
        nowbest += dis(u, p[i]);
    }
    while (step > 1e-10) {
        for (int i = -2; i <= 2; i++) {
            for (int j = -2; j <= 2; j++) {
                v.x = u.x + step * i;
                v.y = u.y + step * j;
                double now = 0;
                for (int i = 0; i < m; ++i) {
                    now += dis(v, p[i]);
                }
                if (now < nowbest) {
                    nowbest = now;
                    u = v;
                }
            }
```

```
        }
        step /= 1.1;    // 会影响 WA || AC || TLE
    }
    return nowbest;
}
```

多边形切割

```
//多边形切割
//可用于半平面交
const int MAXN = 100;
const double EPS = 1e-8;

struct Point {
    double x, y;
};

inline bool zero(double x) {
    return ((x > 0 ? x : -x) < EPS);
}

inline double xmult(const Point & p1, const Point & p2, const Point & p0) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}

bool sameSide(const Point & p1, const Point & p2, const Point & l1, const Point & l
2) {
    return xmult(l1, p1, l2) * xmult(l1, p2, l2) > EPS;
}

Point intersection(const Point & u1, const Point & u2, const Point & v1, const Poin
t & v2) {
    Point ret = u1;
    double t = ((u1.x - v1.x) * (v1.y - v2.y) - (u1.y - v1.y) * (v1.x - v2.x)) / ((
u1.x - u2.x) * (v1.y - v2.y) - (u1.y - u2.y) * (v1.x - v2.x));
    ret.x += (u2.x - u1.x) * t;
    ret.y += (u2.y - u1.y) * t;
    return ret;
}

//将多边形沿 l1, l2 确定的直线切割在 side 侧切割, 保证 l1, l2, side 不共线
void polygonCut(int & n, Point * p, const Point & l1, const Point & l2, const Point
 & side) {
    Point pp[MAXN];
    int m = 0, i;
    for (i = 0; i < n; i++) {
        if (sameSide(p[i], side, l1, l2)) {
            pp[m++] = p[i];
```

```
        }
        if (!sameSide(p[i], p[(i + 1) % n], l1, l2) && !(zero(xmult(p[i], l1, l2))
&& zero(xmult(p[(i + 1) %n ], l1, l2)))) {
            pp[m++] = intersection(p[i], p[(i + 1) % n], l1, l2);
        }
    }
    for (n = i = 0; i < m; i++) {
        if (!i || !zero(pp[i].x - pp[i - 1].x) || !zero(pp[i].y - pp[i-1].y)) {
            p[n++] = pp[i];
        }
    }
    if (zero(p[n - 1].x - p[0].x) && zero(p[n - 1].y - p[0].y)) {
        n--;
    }
    if (n < 3) {
        n = 0;
    }
}
```

浮点函数

```
//浮点几何函数库
#include <cmath>

const double EPS = 1e-8;

struct Point {
    double x, y;
};
struct Line {
    Point a, b;
};

inline bool zero(double x) {
    return ((x > 0 ? x : -x) < EPS);
}

//计算cross product (P1 - P0)x(P2 - P0)
inline double xmult(const Point & p1, const Point & p2, const Point & p0) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}
inline double xmult(double x1, double y1, double x2, double y2, double x0, double y
0) {
    return (x1 - x0) * (y2 - y0) - (x2 - x0) * (y1 - y0);
}

//计算dot product (P1 - P0).(P2 - P0)
inline double dmult(const Point & p1, const Point & p2, const Point & p0) {
```

```
    return (p1.x - p0.x) * (p2.x - p0.x) + (p1.y - p0.y) * (p2.y - p0.y);
}
inline double dmult(double x1, double y1, double x2, double y2, double x0, double y
0) {
    return (x1 - x0) * (x2 - x0) + (y1 - y0) * (y2 - y0);
}

//两点距离
inline double dis(const Point & p1, const Point & p2) {
    return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
}
inline double dis(double x1, double y1, double x2, double y2) {
    return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}

//判三点共线
bool dotsInline(const Point & p1, const Point & p2, const Point & p3) {
    return zero(xmult(p1, p2, p3));
}
bool dotsInline(double x1, double y1, double x2, double y2, double x3, double y3) {
    return zero(xmult(x1, y1, x2, y2, x3, y3));
}

//判点是否在线段上，包括端点
bool dotOnlineIn(const Point & p, const Line & l) {
    return zero(xmult(p, l.a, l.b)) && (l.a.x - p.x) * (l.b.x - p.x) < EPS && (l.a.
y - p.y) * (l.b.y - p.y) < EPS;
}
bool dotOnlineIn(const Point & p, const Point & l1, const Point & l2) {
    return zero(xmult(p, l1, l2)) && (l1.x - p.x) * (l2.x - p.x) < EPS && (l1.y - p
.y) * (l2.y - p.y) < EPS;
}
bool dotOnlineIn(double x, double y, double x1, double y1, double x2, double y2) {
    return zero(xmult(x, y, x1, y1, x2, y2)) && (x1 - x) * (x2 - x) < EPS && (y1 -
y) * (y2 - y) < EPS;
}

//判点是否在线段上，不包括端点
bool dotOnlineEx(const Point & p, const Line & l) {
    return dotOnlineIn(p, l) && (!zero(p.x - l.a.x) || !zero(p.y - l.a.y)) && (!zer
o(p.x - l.b.x) || !zero(p.y - l.b.y));
}
bool dotOnlineEx(const Point & p, const Point & l1, const Point & l2) {
    return dotOnlineIn(p, l1, l2) && (!zero(p.x - l1.x) || !zero(p.y - l1.y)) && (!
zero(p.x - l2.x) || !zero(p.y - l2.y));
}
bool dotOnlineEx(double x, double y, double x1, double y1, double x2, double y2) {
```

```
    return dotOnlineIn(x, y, x1, y1, x2, y2) && (!zero(x - x1) || !zero(y - y1)) &&
 (!zero(x - x2) || !zero(y - y2));
}

//判两点在线段同侧，点在线段上返回0
bool sameSide(const Point & p1, const Point & p2, const Line & l) {
    return xmult(l.a, p1, l.b) * xmult(l.a, p2, l.b) > EPS;
}
bool sameSide(const Point & p1, const Point & p2, const Point & l1, const Point & l
2) {
    return xmult(l1, p1, l2) * xmult(l1, p2, l2) > EPS;
}

//判两点在线段异侧，点在线段上返回0
bool oppositeSide(const Point & p1, const Point & p2, const Line & l) {
    return xmult(l.a, p1, l.b) * xmult(l.a, p2, l.b) < -EPS;
}
bool oppositeSide(const Point & p1, const Point & p2, const Point & l1, const Point
 & l2) {
    return xmult(l1, p1, l2) * xmult(l1, p2, l2) < -EPS;
}

//判两直线平行
bool parallel(const Line & u, const Line & v) {
    return zero((u.a.x - u.b.x) * (v.a.y - v.b.y) - (v.a.x - v.b.x) * (u.a.y - u.b.
y));
}
bool parallel(const Point & u1, const Point & u2, const Point & v1, const Point & v
2) {
    return zero((u1.x - u2.x) * (v1.y - v2.y) - (v1.x - v2.x) * (u1.y - u2.y));
}

//判两直线垂直
bool perpendicular(const Line & u, const Line & v) {
    return zero((u.a.x - u.b.x) * (v.a.x - v.b.x) + (u.a.y - u.b.y) * (v.a.y - v.b.
y));
}
bool perpendicular(const Point & u1, const Point & u2, const Point & v1, const Poin
t & v2) {
    return zero((u1.x - u2.x) * (v1.x - v2.x) + (u1.y - u2.y) * (v1.y - v2.y));
}

//判两线段相交，包括端点和部分重合
bool intersectIn(const Line & u, const Line & v) {
    if (!dotsInline(u.a, u.b, v.a) || !dotsInline(u.a, u.b, v.b)) {
        return !sameSide(u.a, u.b, v) && !sameSide(v.a, v.b, u);
    } else {
```

```
        return dotOnlineIn(u.a, v) || dotOnlineIn(u.b, v) || dotOnlineIn(v.a, u) ||
 dotOnlineIn(v.b, u);
    }
}
bool intersectIn(const Point & u1, const Point & u2, const Point & v1, const Point
& v2) {
    if (!dotsInline(u1, u2, v1) || !dotsInline(u1, u2, v2)) {
        return !sameSide(u1, u2, v1, v2) && !sameSide(v1, v2, u1, u2);
    } else {
        return dotOnlineIn(u1, v1, v2) || dotOnlineIn(u2, v1, v2) || dotOnlineIn(v1
, u1, u2) || dotOnlineIn(v2, u1, u2);
    }
}

//判两线段相交，不包括端点和部分重合
bool intersectEx(const Line & u, const Line & v) {
    return oppositeSide(u.a, u.b, v) && oppositeSide(v.a, v.b, u);
}
bool intersectEx(const Point & u1, const Point & u2, const Point & v1, const Point
& v2) {
    return oppositeSide(u1, u2, v1, v2) && oppositeSide(v1, v2, u1, u2);
}

//计算两直线交点，注意事先判断直线是否平行!
//线段交点请另外判线段相交(同时还是要判断是否平行!)
Point intersection(const Line & u, const Line & v) {
    Point ret = u.a;
    double t = ((u.a.x - v.a.x) * (v.a.y - v.b.y) - (u.a.y - v.a.y) * (v.a.x - v.b.
x)) / ((u.a.x - u.b.x) * (v.a.y - v.b.y) - (u.a.y - u.b.y) * (v.a.x - v.b.x));
    ret.x += (u.b.x - u.a.x) * t;
    ret.y += (u.b.y - u.a.y) * t;
    return ret;
}
Point intersection(const Point & u1, const Point & u2, const Point & v1, const Poin
t & v2) {
    Point ret = u1;
    double t = ((u1.x - v1.x) * (v1.y - v2.y) - (u1.y - v1.y) * (v1.x - v2.x)) / ((
u1.x - u2.x) * (v1.y - v2.y) - (u1.y - u2.y) * (v1.x - v2.x));
    ret.x += (u2.x - u1.x) * t;
    ret.y += (u2.y - u1.y) * t;
    return ret;
}

//点到直线上的最近点
Point ptoline(const Point & p, const Line & l) {
    Point t = p;
    t.x += l.a.y - l.b.y;
```

```
    t.y += l.b.x - l.a.x;
    return intersection(p, t, l.a, l.b);
}
Point ptoline(const Point & p, const Point & l1, const Point & l2) {
    Point t = p;
    t.x += l1.y - l2.y;
    t.y += l2.x - l1.x;
    return intersection(p, t, l1, l2);
}

//点到直线距离
double disptoline(const Point & p, const Line & l) {
    return fabs(xmult(p, l.a, l.b)) / dis(l.a, l.b);
}
double disptoline(const Point & p, const Point & l1, const Point & l2) {
    return fabs(xmult(p, l1, l2)) / dis(l1, l2);
}
double disptoline(double x, double y, double x1, double y1, double x2, double y2) {
    return fabs(xmult(x, y, x1, y1, x2, y2)) / dis(x1, y1, x2, y2);
}

//点到线段上的最近点
Point ptoseg(const Point & p, const Line & l) {
    Point t = p;
    t.x += l.a.y - l.b.y;
    t.y += l.b.x - l.a.x;
    if (xmult(l.a, t, p) * xmult(l.b, t, p) > EPS) {
        return dis(p, l.a) < dis(p, l.b) ? l.a : l.b;
    } else {
        return intersection(p, t, l.a, l.b);
    }
}
Point ptoseg(const Point & p, const Point & l1, const Point & l2) {
    Point t = p;
    t.x += l1.y - l2.y;
    t.y += l2.x - l1.x;
    if (xmult(l1, t, p) * xmult(l2, t, p) > EPS) {
        return dis(p, l1) < dis(p, l2) ? l1 : l2;
    } else {
        return intersection(p, t, l1, l2);
    }
}

//点到线段距离
double disptoseg(const Point & p, const Line & l) {
    Point t = p;
    t.x += l.a.y - l.b.y;
```

```
    t.y += l.b.x - l.a.x;
    if (xmult(l.a, t, p) * xmult(l.b, t, p) > EPS) {
        return dis(p, l.a) < dis(p, l.b) ? dis(p, l.a) : dis(p, l.b);
    } else {
        return fabs(xmult(p, l.a, l.b))/dis(l.a, l.b);
    }
}
double disptoseg(const Point & p, const Point & l1, const Point & l2) {
    Point t = p;
    t.x += l1.y - l2.y;
    t.y += l2.x - l1.x;
    if (xmult(l1, t, p) * xmult(l2, t, p) > EPS) {
        return dis(p, l1) < dis(p, l2) ? dis(p, l1) : dis(p, l2);
    } else {
        return fabs(xmult(p, l1, l2))/dis(l1, l2);
    }
}

//矢量 v 以 p 为顶点逆时针旋转 angle 并放大 scale 倍
Point rotate(Point v, Point p, double angle, double scale) {
    Point ret = p;
    v.x -= p.x;
    v.y -= p.y;
    p.x = scale * cos(angle);
    p.y = scale * sin(angle);
    ret.x += v.x * p.x - v.y * p.y;
    ret.y += v.x * p.y + v.y * p.x;
    return ret;
}
```

面积

```
#include <cmath>

struct Point {
    double x, y;
};

//计算 cross product (P1-P0)x(P2-P0)
inline double xmult(const Point & p1, const Point & p2, const Point & p0) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}
inline double xmult(double x1, double y1, double x2, double y2, double x0, double y
0) {
    return (x1 - x0) * (y2 - y0) - (x2 - x0) * (y1 - y0);
}

//计算三角形面积, 输入三顶点
```

```
double areaTriangle(const Point & p1, const Point & p2, const Point & p3) {
    return fabs(xmult(p1, p2, p3)) / 2;
}
double areaTriangle(double x1, double y1, double x2, double y2, double x3, double y
3) {
    return fabs(xmult(x1, y1, x2, y2, x3, y3)) / 2;
}

//计算三角形面积, 输入三边长
double areaTriangle(double a, double b, double c) {
    double s = (a + b + c) / 2;
    return sqrt(s * (s - a) * (s - b) * (s - c));
}

//计算多边形面积, 顶点按顺时针或逆时针给出
double areaPolygon(int n, const Point* p) {
    double s1 = 0, s2 = 0;
    for (int i = 0; i < n; i++) {
        s1 += p[(i + 1) % n].y * p[i].x;
        s2 += p[(i + 1) % n].y * p[(i + 2) % n].x;
    }
    return fabs(s1 - s2)/2;
}
```

球面

```
#include <cmath>

const double PI = acos(-1.0);

//计算圆心角 lat 表示纬度, -90 <= w <= 90, lng 表示经度
//返回两点所在大圆劣弧对应圆心角, 0 <= angle <= PI
double angle(double lng1, double lat1, double lng2, double lat2) {
    double dlng = fabs(lng1 - lng2) * PI / 180;
    while (dlng >= PI + PI) {
        dlng -= PI + PI;
    }
    if (dlng > PI) {
        dlng = PI + PI - dlng;
    }
    lat1 *= PI / 180;
    lat2 *= PI / 180;
    return acos(cos(lat1) * cos(lat2) * cos(dlng) + sin(lat1) * sin(lat2));
}

//计算距离, r 为球半径
double lineDist(double r, double lng1, double lat1, double lng2, double lat2) {
    double dlng = fabs(lng1 - lng2) * PI / 180;
```

```
    while (dlng >= PI + PI) {
        dlng -= PI + PI;
    }
    if (dlng > PI) {
        dlng = PI + PI - dlng;
    }
    lat1 *= PI / 180;
    lat2 *= PI / 180;
    return r * sqrt(2 - 2 * (cos(lat1) * cos(lat2) * cos(dlng) + sin(lat1) * sin(la
t2)));
}

//计算球面距离, r 为球半径
inline double sphereDist(double r, double lng1, double lat1, double lng2, double la
t2) {
    return r * angle(lng1, lat1, lng2, lat2);
}
```

三角形

```
#include <cmath>

struct Point {
    double x, y;
};
struct Line {
    Point a, b;
};

inline double dis(const Point & p1, const Point & p2) {
    return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
}

Point intersection(const Line & u, const Line & v) {
    Point ret = u.a;
    double t = ((u.a.x - v.a.x) * (v.a.y - v.b.y) - (u.a.y - v.a.y) * (v.a.x - v.b.
x)) / ((u.a.x - u.b.x) * (v.a.y - v.b.y) - (u.a.y - u.b.y) * (v.a.x - v.b.x));
    ret.x += (u.b.x - u.a.x) * t;
    ret.y += (u.b.y - u.a.y) * t;
    return ret;
}

//外心
Point circumcenter(const Point & a, const Point & b, const Point & c) {
    Line u, v;
    u.a.x = (a.x + b.x) / 2;
    u.a.y = (a.y + b.y) / 2;
    u.b.x = u.a.x - a.y + b.y;
```

```
    u.b.y = u.a.y + a.x - b.x;
    v.a.x = (a.x + c.x) / 2;
    v.a.y = (a.y + c.y) / 2;
    v.b.x = v.a.x - a.y + c.y;
    v.b.y = v.a.y + a.x - c.x;
    return intersection(u, v);
}

//内心
Point incenter(const Point & a, const Point & b, const Point & c) {
    Line u, v;
    double m, n;
    u.a = a;
    m = atan2(b.y - a.y, b.x - a.x);
    n = atan2(c.y - a.y, c.x - a.x);
    u.b.x = u.a.x + cos((m + n) / 2);
    u.b.y = u.a.y + sin((m + n) / 2);
    v.a = b;
    m = atan2(a.y - b.y, a.x - b.x);
    n = atan2(c.y - b.y, c.x - b.x);
    v.b.x = v.a.x + cos((m + n) / 2);
    v.b.y = v.a.y + sin((m + n) / 2);
    return intersection(u, v);
}

//垂心
Point perpencenter(const Point & a, const Point & b, const Point & c) {
    Line u, v;
    u.a = c;
    u.b.x = u.a.x - a.y + b.y;
    u.b.y = u.a.y + a.x - b.x;
    v.a = b;
    v.b.x = v.a.x - a.y + c.y;
    v.b.y = v.a.y + a.x - c.x;
    return intersection(u, v);
}

//重心
//到三角形三顶点距离的平方和最小的点
//三角形内到三边距离之积最大的点
Point barycenter(const Point & a, const Point & b, const Point & c) {
    Line u, v;
    u.a.x = (a.x + b.x) / 2;
    u.a.y = (a.y + b.y) / 2;
    u.b = c;
    v.a.x = (a.x + c.x) / 2;
    v.a.y = (a.y + c.y) / 2;
```

```
    v.b = b;
    return intersection(u, v);
}

//费马点
//到三角形三顶点距离之和最小的点
/*
Point fermentpoint(const Point & a, const Point & b, const Point & c) {
    Point u, v;
    double step = fabs(a.x) + fabs(a.y) + fabs(b.x) + fabs(b.y) + fabs(c.x) + fabs(
c.y);
    u.x = (a.x + b.x + c.x) / 3;
    u.y = (a.y + b.y + c.y) / 3;
    while (step > 1e-10) {
        for (int k = 0; k < 10; step /= 2, k++) {
            for (int i = -1; i <= 1; i++) {
                for (int j = -1; j <= 1; j++) {
                    v.x = u.x + step * i;
                    v.y = u.y + step * j;
                    if (dis(u, a) + dis(u, b) + dis(u, c) > dis(v, a) + dis(v, b) +
 dis(v, c)) {
                        u = v;
                    }
                }
            }
        }
    }
    return u;
}
*/

// 可能WA，参考多边形费马点
```

三维几何

```
//三维几何函数库
#include <cmath>

const double EPS = 1e-8;

struct Point3D {
    double x, y, z;
};

struct Line3D {
    Point3D a, b;
};
```

```
struct Plane {
    Point3D a, b, c;
};

struct PlaneF {
    // ax + by + cz + d = 0
    double a, b, c, d;
};

inline bool zero(double x) {
    return (x > 0 ? x : -x) < EPS;
}

//平方
inline double sqr(double d) {
    return d * d;
}

//计算cross product U x V
inline Point3D xmult(const Point3D & u, const Point3D & v) {
    Point3D ret;
    ret.x = u.y * v.z - v.y * u.z;
    ret.y = u.z * v.x - u.x * v.z;
    ret.z = u.x * v.y - u.y * v.x;
    return ret;
}

//计算dot product U . V
inline double dmult(const Point3D & u, const Point3D & v) {
    return u.x * v.x + u.y * v.y + u.z * v.z;
}

//矢量差U - V
inline Point3D subt(const Point3D & u, const Point3D & v) {
    Point3D ret;
    ret.x = u.x - v.x;
    ret.y = u.y - v.y;
    ret.z = u.z - v.z;
    return ret;
}

//取平面法向量
inline Point3D pvec(const Plane & s) {
    return xmult(subt(s.a, s.b), subt(s.b, s.c));
}
inline Point3D pvec(const Point3D & s1, const Point3D & s2, const Point3D & s3) {
    return xmult(subt(s1, s2), subt(s2, s3));
```

```
}
inline Point3D pvec(const PlaneF & p) {
    Point3D ret;
    ret.x = p.a;
    ret.y = p.b;
    ret.z = p.c;
    return ret;
}

//两点距离
inline double dis(const Point3D & p1, const Point3D & p2) {
    return sqrt((p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y) + (p1.z -
 p2.z)*(p1.z - p2.z));
}

//向量大小
inline double vlen(const Point3D & p) {
    return sqrt(p.x*p.x + p.y*p.y + p.z*p.z);
}

//向量大小的平方
inline double sqrlen(const Point3D & p) {
    return (p.x*p.x + p.y*p.y + p.z*p.z);
}

//判三点共线
bool dotsInline(const Point3D & p1, const Point3D & p2, const Point3D & p3) {
    return sqrlen(xmult(subt(p1, p2), subt(p2, p3))) < EPS;
}

//判四点共面
bool dotsOnplane(const Point3D & a, const Point3D & b, const Point3D & c, const Poi
nt3D & d) {
    return zero(dmult(pvec(a, b, c), subt(d, a)));
}

//判点是否在线段上, 包括端点和共线
bool dotOnlineIn(const Point3D & p, const Line3D & l) {
    return zero(sqrlen(xmult(subt(p, l.a), subt(p, l.b)))) && (l.a.x - p.x) * (l.b.
x - p.x) < EPS && (l.a.y - p.y) * (l.b.y - p.y) < EPS && (l.a.z - p.z) * (l.b.z - p
.z) < EPS;
}
bool dotOnlineIn(const Point3D & p, const Point3D & l1, const Point3D & l2) {
    return zero(sqrlen(xmult(subt(p, l1), subt(p, l2)))) && (l1.x - p.x) * (l2.x -
p.x) < EPS && (l1.y - p.y) * (l2.y - p.y) < EPS && (l1.z - p.z) * (l2.z - p.z) < EP
S;
}
```

```
//判点是否在线段上，不包括端点
bool dotOnlineEx(const Point3D & p, const Line3D & l) {
    return dotOnlineIn(p, l) && (!zero(p.x - l.a.x) || !zero(p.y - l.a.y) || !zero(
p.z - l.a.z)) && (!zero(p.x - l.b.x) || !zero(p.y - l.b.y) || !zero(p.z - l.b.z));
}
bool dotOnlineEx(const Point3D & p, const Point3D & l1, const Point3D & l2) {
    return dotOnlineIn(p, l1, l2) && (!zero(p.x - l1.x) || !zero(p.y - l1.y) || !ze
ro(p.z - l1.z)) && (!zero(p.x - l2.x) || !zero(p.y - l2.y) || !zero(p.z - l2.z));
}

//判点是否在空间三角形上，包括边界，三点共线无意义
bool dotInplaneIn(const Point3D & p, const Plane & s) {
    return zero(vlen(xmult(subt(s.a, s.b), subt(s.a, s.c))) - vlen(xmult(subt(p, s.
a), subt(p, s.b))) - vlen(xmult(subt(p, s.b), subt(p, s.c))) - vlen(xmult(subt(p, s
.c), subt(p, s.a))));
}
bool dotInplaneIn(const Point3D & p, const Point3D & s1, const Point3D & s2, const
Point3D & s3) {
    return zero(vlen(xmult(subt(s1, s2), subt(s1, s3))) - vlen(xmult(subt(p, s1), s
ubt(p, s2))) - vlen(xmult(subt(p, s2), subt(p, s3))) - vlen(xmult(subt(p, s3), subt
(p, s1))));
}

//判点是否在空间三角形上，不包括边界，三点共线无意义
bool dotInplaneEx(const Point3D & p, const Plane & s) {
    return dotInplaneIn(p, s) && sqrlen(xmult(subt(p, s.a), subt(p, s.b))) > EPS &&
 sqrlen(xmult(subt(p, s.b), subt(p, s.c))) > EPS && sqrlen(xmult(subt(p, s.c), subt
(p, s.a))) > EPS;
}
bool dotInplaneEx(const Point3D & p, const Point3D & s1, const Point3D & s2, const
Point3D & s3) {
    return dotInplaneIn(p, s1, s2, s3) && sqrlen(xmult(subt(p, s1), subt(p, s2))) >
 EPS && sqrlen(xmult(subt(p, s2), subt(p, s3))) > EPS && sqrlen(xmult(subt(p, s3),
subt(p, s1))) > EPS;
}

//判两点在线段同侧，点在线段上返回 0，不共面无意义
bool sameSide(const Point3D & p1, const Point3D & p2, const Line3D & l) {
    return dmult(xmult(subt(l.a, l.b), subt(p1, l.b)), xmult(subt(l.a, l.b), subt(p
2, l.b))) > EPS;
}
bool sameSide(const Point3D & p1, const Point3D & p2, const Point3D & l1, const Poi
nt3D & l2) {
    return dmult(xmult(subt(l1, l2), subt(p1, l2)), xmult(subt(l1, l2), subt(p2, l2
))) > EPS;
}
```

```
//判两点在线段异侧，点在线段上返回0，不共面无意义
bool oppositeSide(const Point3D & p1, const Point3D & p2, const Line3D & l) {
    return dmult(xmult(subt(l.a, l.b), subt(p1, l.b)), xmult(subt(l.a, l.b), subt(p
2, l.b))) < -EPS;
}
bool oppositeSide(const Point3D & p1, const Point3D & p2, const Point3D & l1, const
 Point3D & l2) {
    return dmult(xmult(subt(l1, l2), subt(p1, l2)), xmult(subt(l1, l2), subt(p2, l2
))) < -EPS;
}

//判两点在平面同侧，点在平面上返回0
bool sameSide(const Point3D & p1, const Point3D & p2, const Plane & s) {
    return dmult(pvec(s), subt(p1, s.a)) * dmult(pvec(s), subt(p2, s.a)) > EPS;
}
bool sameSide(const Point3D & p1, const Point3D & p2, const Point3D & s1, const Poi
nt3D & s2, const Point3D & s3) {
    return dmult(pvec(s1, s2, s3), subt(p1, s1)) * dmult(pvec(s1, s2, s3), subt(p2,
 s1)) > EPS;
}
bool sameSide(const Point3D & p1, const Point3D & p2, const PlaneF & s) {
    return (s.a * p1.x + s.b * p1.y + s.c * p1.z + s.d) * (s.a * p2.x + s.b * p2.y
+ s.c * p2.z + s.d) > EPS;
}

//判两点在平面异侧，点在平面上返回0
bool oppositeSide(const Point3D & p1, const Point3D & p2, const Plane & s) {
    return dmult(pvec(s), subt(p1, s.a)) * dmult(pvec(s), subt(p2, s.a)) < -EPS;
}
bool oppositeSide(const Point3D & p1, const Point3D & p2, const Point3D & s1, const
 Point3D & s2, const Point3D & s3) {
    return dmult(pvec(s1, s2, s3), subt(p1, s1)) * dmult(pvec(s1, s2, s3), subt(p2,
 s1)) < -EPS;
}
bool oppositeSide(const Point3D & p1, const Point3D & p2, const PlaneF & s) {
    return (s.a*p1.x+s.b*p1.y+s.c*p1.z+s.d) * (s.a*p2.x+s.b*p2.y+s.c*p2.z+s.d) < -E
PS;
}

//判两直线平行
bool parallel(const Line3D & u, const Line3D & v) {
    return sqrlen(xmult(subt(u.a, u.b), subt(v.a, v.b))) < EPS;
}
bool parallel(const Point3D & u1, const Point3D & u2, const Point3D & v1, const Poi
nt3D & v2) {
    return sqrlen(xmult(subt(u1, u2), subt(v1, v2))) < EPS;
```

```
}

//判两平面平行
bool parallel(const Plane & u, const Plane & v) {
    return sqrlen(xmult(pvec(u), pvec(v))) < EPS;
}
bool parallel(const Point3D & u1, const Point3D & u2, const Point3D & u3, const Poi
nt3D & v1, const Point3D & v2, const Point3D & v3) {
    return sqrlen(xmult(pvec(u1, u2, u3), pvec(v1, v2, v3))) < EPS;
}
bool parallel(const PlaneF & u, const PlaneF & v) {
    return sqrlen(xmult(pvec(u), pvec(v))) < EPS;
}

//判直线与平面平行
bool parallel(const Line3D & l, const Plane & s) {
    return zero(dmult(subt(l.a, l.b), pvec(s)));
}
bool parallel(const Point3D & l1, const Point3D & l2, const Point3D & s1, const Poi
nt3D & s2, const Point3D & s3) {
    return zero(dmult(subt(l1, l2), pvec(s1, s2, s3)));
}
bool parallel(const Line3D & l, const PlaneF & s) {
    return zero(dmult(subt(l.a, l.b), pvec(s)));
}

//判两直线垂直
bool perpendicular(const Line3D & u, const Line3D & v) {
    return zero(dmult(subt(u.a, u.b), subt(v.a, v.b)));
}
bool perpendicular(const Point3D & u1, const Point3D & u2, const Point3D & v1, cons
t Point3D & v2) {
    return zero(dmult(subt(u1, u2), subt(v1, v2)));
}

//判两平面垂直
bool perpendicular(const Plane & u, const Plane & v) {
    return zero(dmult(pvec(u), pvec(v)));
}
bool perpendicular(const Point3D & u1, const Point3D & u2, const Point3D & u3, cons
t Point3D & v1, const Point3D & v2, const Point3D & v3) {
    return zero(dmult(pvec(u1, u2, u3), pvec(v1, v2, v3)));
}
bool perpendicular(const PlaneF & u, const PlaneF & v) {
    return zero(dmult(pvec(u), pvec(v)));
}
```

```
//判直线与平面垂直
bool perpendicular(const Line3D & l, const Plane & s) {
    return sqrlen(xmult(subt(l.a, l.b), pvec(s))) < EPS;
}
bool perpendicular(const Point3D & l1, const Point3D & l2, const Point3D & s1, cons
t Point3D & s2, const Point3D & s3) {
    return sqrlen(xmult(subt(l1, l2), pvec(s1, s2, s3))) < EPS;
}
bool perpendicular(const Line3D & l, const PlaneF & s) {
    return sqrlen(xmult(subt(l.a, l.b), pvec(s))) < EPS;
}

//判两线段相交，包括端点和部分重合
bool intersectIn(const Line3D & u, const Line3D & v) {
    if (!dotsOnplane(u.a, u.b, v.a, v.b)) {
        return 0;
    } else if (!dotsInline(u.a, u.b, v.a) || !dotsInline(u.a, u.b, v.b)) {
        return !sameSide(u.a, u.b, v) && !sameSide(v.a, v.b, u);
    } else {
        return dotOnlineIn(u.a, v) || dotOnlineIn(u.b, v) || dotOnlineIn(v.a, u) ||
 dotOnlineIn(v.b, u);
    }
}
bool intersectIn(const Point3D & u1, const Point3D & u2, const Point3D & v1, const
Point3D & v2) {
    if (!dotsOnplane(u1, u2, v1, v2)) {
        return 0;
    } else if (!dotsInline(u1, u2, v1) || !dotsInline(u1, u2, v2)) {
        return !sameSide(u1, u2, v1, v2) && !sameSide(v1, v2, u1, u2);
    } else {
        return dotOnlineIn(u1, v1, v2) || dotOnlineIn(u2, v1, v2) || dotOnlineIn(v1
, u1, u2) || dotOnlineIn(v2, u1, u2);
    }
}

//判两线段相交，不包括端点和部分重合
bool intersectEx(const Line3D & u, const Line3D & v) {
    return dotsOnplane(u.a, u.b, v.a, v.b) && oppositeSide(u.a, u.b, v) && opposite
Side(v.a, v.b, u);
}
bool intersectEx(const Point3D & u1, const Point3D & u2, const Point3D & v1, const
Point3D & v2) {
    return dotsOnplane(u1, u2, v1, v2) && oppositeSide(u1, u2, v1, v2) && oppositeS
ide(v1, v2, u1, u2);
}

//判线段与空间三角形相交，包括交于边界和(部分)包含
```

```
bool intersectIn(const Line3D & l, const Plane & s) {
    return !sameSide(l.a, l.b, s) && !sameSide(s.a, s.b, l.a, l.b, s.c) && !sameSid
e(s.b, s.c, l.a, l.b, s.a) && !sameSide(s.c, s.a, l.a, l.b, s.b);
}
bool intersectIn(const Point3D & l1, const Point3D & l2, const Point3D & s1, const
Point3D & s2, const Point3D & s3) {
    return !sameSide(l1, l2, s1, s2, s3) && !sameSide(s1, s2, l1, l2, s3) && !sameS
ide(s2, s3, l1, l2, s1) && !sameSide(s3, s1, l1, l2, s2);
}

//判线段与空间三角形相交，不包括交于边界和(部分)包含
bool intersectEx(const Line3D & l, const Plane & s) {
    return oppositeSide(l.a, l.b, s) && oppositeSide(s.a, s.b, l.a, l.b, s.c) && op
positeSide(s.b, s.c, l.a, l.b, s.a) && oppositeSide(s.c, s.a, l.a, l.b, s.b);
}
bool intersectEx(const Point3D & l1, const Point3D & l2, const Point3D & s1, const
Point3D & s2, const Point3D & s3) {
    return oppositeSide(l1, l2, s1, s2, s3) && oppositeSide(s1, s2, l1, l2, s3) &&
oppositeSide(s2, s3, l1, l2, s1) && oppositeSide(s3, s1, l1, l2, s2);
}

//计算两直线交点，注意事先判断直线是否共面和平行!
//线段交点请另外判线段相交(同时还是要判断是否平行!)
#include <algorithm>

using namespace std;
/*
Point3D intersection(Line3D u, Line3D v) {
    if (zero(u.a.x - u.b.x) && zero(v.a.x - v.b.x)) {
        swap(u.a.x, u.a.z);
        swap(u.b.x, u.b.z);
        swap(v.a.x, v.a.z);
        swap(v.b.x, v.b.z);
        Point3D ret = intersection(u, v);
        swap(ret.x, ret.z);
        return ret;
    } else if (zero(u.a.y - u.b.y) && zero(v.a.y - v.b.y)) {
        swap(u.a.y, u.a.z);
        swap(u.b.y, u.b.z);
        swap(v.a.y, v.a.z);
        swap(v.b.y, v.b.z);
        Point3D ret = intersection(u, v);
        swap(ret.y, ret.z);
        return ret;
    } else {
        Point3D ret = u.a;
        double t = ((u.a.x - v.a.x) * (v.a.y - v.b.y) - (u.a.y - v.a.y) * (v.a.x -
```

```
v.b.x)) / ((u.a.x - u.b.x) * (v.a.y - v.b.y) - (u.a.y - u.b.y) * (v.a.x - v.b.x));
        ret.x += (u.b.x - u.a.x) * t;
        ret.y += (u.b.y - u.a.y) * t;
        ret.z += (u.b.z - u.a.z) * t;
        return ret;
    }
}*/
Point3D intersection(Point3D u1, Point3D u2, Point3D v1, Point3D v2) {
   double dxu = u2.x - u1.x;
   double dyu = u2.y - u1.y;
   double dzu = u2.z - u1.z;
   double dxv = v2.x - v1.x;
   double dyv = v2.y - v1.y;
   double dzv = v2.z - v1.z;
   double t;
   if (!zero(dxu * dyv - dyu * dxv)) {
      t = (dyv * (v1.x - u1.x) + dxv * (u1.y - v1.y)) / (dxu * dyv - dyu * dxv);
   } else if (!zero(dxu * dzv - dzu * dxv)) {
      t = (dzv * (v1.x - u1.x) + dxv * (u1.z - v1.z)) / (dxu * dzv - dzu * dxv);
   } else {
      t = (dzv * (v1.y - u1.y) + dyv * (u1.z - v1.z)) / (dyu * dzv - dzu * dyv);
   }
   Point3D ret;
   ret.x = u1.x + dxu * t;
   ret.y = u1.y + dyu * t;
   ret.z = u1.z + dzu * t;
   return ret;
}

//计算直线与平面交点，注意事先判断是否平行，并保证三点不共线!
//线段和空间三角形交点请另外判断
Point3D intersection(const Line3D & l, const Plane & s) {
    Point3D ret = pvec(s);
    double t = (ret.x * (s.a.x - l.a.x) + ret.y * (s.a.y - l.a.y) + ret.z * (s.a.z
- l.a.z)) / (ret.x * (l.b.x - l.a.x) + ret.y * (l.b.y - l.a.y) + ret.z * (l.b.z - l
.a.z));
    ret.x = l.a.x + (l.b.x - l.a.x) * t;
    ret.y = l.a.y + (l.b.y - l.a.y) * t;
    ret.z = l.a.z + (l.b.z - l.a.z) * t;
    return ret;
}
Point3D intersection(const Point3D & l1, const Point3D & l2, const Point3D & s1, co
nst Point3D & s2, const Point3D & s3) {
    Point3D ret = pvec(s1, s2, s3);
    double t = (ret.x * (s1.x - l1.x) + ret.y * (s1.y - l1.y) + ret.z * (s1.z - l1.
z)) / (ret.x * (l2.x - l1.x) + ret.y * (l2.y - l1.y) + ret.z * (l2.z - l1.z));
    ret.x = l1.x + (l2.x - l1.x) * t;
```

```
    ret.y = l1.y + (l2.y - l1.y) * t;
    ret.z = l1.z + (l2.z - l1.z) * t;
    return ret;
}
Point3D intersection(const Line3D & l, const PlaneF & s) {
    Point3D ret = subt(l.b, l.a);
    double t = -(dmult(pvec(s), l.a) + s.d) / (dmult(pvec(s), ret));
    ret.x = ret.x * t + l.a.x;
    ret.y = ret.y * t + l.a.y;
    ret.z = ret.z * t + l.a.z;
    return ret;
}

//计算两平面交线，注意事先判断是否平行，并保证三点不共线!
Line3D intersection(const Plane & u, const Plane & v) {
    Line3D ret;
    ret.a = parallel(v.a, v.b, u.a, u.b, u.c) ? intersection(v.b, v.c, u.a, u.b, u.
c) : intersection(v.a, v.b, u.a, u.b, u.c);
    ret.b = parallel(v.c, v.a, u.a, u.b, u.c) ? intersection(v.b, v.c, u.a, u.b, u.
c) : intersection(v.c, v.a, u.a, u.b, u.c);
    return ret;
}
Line3D intersection(const Point3D & u1, const Point3D & u2, const Point3D & u3, con
st Point3D & v1, const Point3D & v2, const Point3D & v3) {
    Line3D ret;
    ret.a = parallel(v1, v2, u1, u2, u3) ? intersection(v2, v3, u1, u2, u3) : inter
section(v1, v2, u1, u2, u3);
    ret.b = parallel(v3, v1, u1, u2, u3) ? intersection(v2, v3, u1, u2, u3) : inter
section(v3, v1, u1, u2, u3);
    return ret;
}

//点到直线距离
double disptoline(const Point3D & p, const Line3D & l) {
    return vlen(xmult(subt(p, l.a), subt(l.b, l.a))) / dis(l.a, l.b);
}
double disptoline(const Point3D & p, const Point3D & l1, const Point3D & l2) {
    return vlen(xmult(subt(p, l1), subt(l2, l1))) / dis(l1, l2);
}

//点到直线最近点
Point3D ptoline(const Point3D & p, const Line3D & l) {
    Point3D ab = subt(l.b, l.a);
    double t = - dmult(subt(p, l.a), ab) / sqrlen(ab);
    ab.x *= t;
    ab.y *= t;
    ab.z *= t;
```

```
    return subt(l.a, ab);
}

//点到平面距离
double disptoplane(const Point3D & p, const Plane & s) {
    return fabs(dmult(pvec(s), subt(p, s.a))) / vlen(pvec(s));
}
double disptoplane(const Point3D & p, const Point3D & s1, const Point3D & s2, const
 Point3D & s3) {
    return fabs(dmult(pvec(s1, s2, s3), subt(p, s1))) / vlen(pvec(s1, s2, s3));
}
double disptoplane(const Point3D & p, const PlaneF & s) {
    return fabs((dmult(pvec(s), p)+s.d) / vlen(pvec(s)));
}

//点到平面最近点
Point3D ptoplane(const Point3D & p, const PlaneF & s) {
    Line3D l;
    l.a = p;
    l.b = pvec(s);
    l.b.x += p.x;
    l.b.y += p.y;
    l.b.z += p.z;
    return intersection(l, s);
}

//直线到直线距离
double dislinetoline(const Line3D & u, const Line3D & v) {
    Point3D n = xmult(subt(u.a, u.b), subt(v.a, v.b));
    return fabs(dmult(subt(u.a, v.a), n)) / vlen(n);
}
double dislinetoline(const Point3D & u1, const Point3D & u2, const Point3D & v1, co
nst Point3D & v2) {
    Point3D n = xmult(subt(u1, u2), subt(v1, v2));
    return fabs(dmult(subt(u1, v1), n)) / vlen(n);
}

//直线到直线的最近点对
//p1 在 u 上，p2 在 v 上，p1 到 p2 是 uv 之间的最近距离
//注意，保证两直线不平行
void linetoline(const Line3D & u, const Line3D & v, Point3D &p1, Point3D &p2) {
    Point3D ab = subt(u.b, u.a), cd = subt(v.b, v.a), ac = subt(v.a, u.a);
    double r = (dmult(ab, cd) * dmult(ac, ab) - sqrlen(ab) * dmult(ac, cd)) / (sqrl
en(ab)*sqrlen(cd) - sqr(dmult(ab, cd)));
    p2.x = v.a.x + r * cd.x;
    p2.y = v.a.y + r * cd.y;
    p2.z = v.a.z + r * cd.z;
```

```
    p1 = ptoline(p2, u);
}

//两直线夹角 cos 值
double angleCos(const Line3D & u, const Line3D & v) {
    return dmult(subt(u.a, u.b), subt(v.a, v.b)) / vlen(subt(u.a, u.b)) / vlen(subt
(v.a, v.b));
}
double angleCos(const Point3D & u1, const Point3D & u2, const Point3D & v1, const P
oint3D & v2) {
    return dmult(subt(u1, u2), subt(v1, v2)) / vlen(subt(u1, u2)) / vlen(subt(v1, v
2));
}

//两平面夹角 cos 值
double angleCos(const Plane & u, const Plane & v) {
    return dmult(pvec(u), pvec(v)) / vlen(pvec(u)) / vlen(pvec(v));
}
double angleCos(const Point3D & u1, const Point3D & u2, const Point3D & u3, const P
oint3D & v1, const Point3D & v2, const Point3D & v3) {
    return dmult(pvec(u1, u2, u3), pvec(v1, v2, v3)) / vlen(pvec(u1, u2, u3)) / vle
n(pvec(v1, v2, v3));
}
double angleCos(const PlaneF & u, const PlaneF & v) {
    return dmult(pvec(u), pvec(v)) / (vlen(pvec(u)) * vlen(pvec(v)));
}

//直线平面夹角 sin 值
double angleSin(const Line3D & l, const Plane & s) {
    return dmult(subt(l.a, l.b), pvec(s)) / vlen(subt(l.a, l.b)) / vlen(pvec(s));
}
double angleSin(const Point3D & l1, const Point3D & l2, const Point3D & s1, const P
oint3D & s2, const Point3D & s3) {
    return dmult(subt(l1, l2), pvec(s1, s2, s3)) / vlen(subt(l1, l2)) / vlen(pvec(s
1, s2, s3));
}
double angleSin(Line3D l, const PlaneF & s) {
    return dmult(subt(l.a, l.b), pvec(s)) / (vlen(subt(l.a, l.b)) * vlen(pvec(s)));
}

// 平面方程形式转化 Plane -> PlaneF
PlaneF planeToPlaneF(const Plane & p) {
    PlaneF ret;
    Point3D m = xmult(subt(p.b, p.a), subt(p.c, p.a));
    ret.a = m.x;
    ret.b = m.y;
    ret.c = m.z;
```

```
    ret.d = -m.x * p.a.x - m.y * p.a.y - m.z * p.a.z;
    return ret;
}
```

凸包

```
// 建议使用CONVEX HULL III by VB

// CONVEX HULL I

#include <cstdlib>

const double EPS = 1e-8;

struct Point {
    double x, y;
};

inline bool zero(double x) {
    return ((x > 0 ? x : -x) < EPS);
}

//计算cross product (P1 - P0)x(P2 - P0)
inline double xmult(const Point & p1, const Point & p2, const Point & p0) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}

//graham算法顺时针构造包含所有共线点的凸包, O(nlogn)
Point p1, p2;
int graham_cp(const void * a, const void * b) {
    double ret = xmult(*((Point *)a), *((Point *)b), p1);
    return zero(ret) ? (xmult(*((Point*)a), ((Point)b), p2) > 0 ? 1 : -1) : (ret >
0 ? 1 : -1);
}
void _graham(int n, Point* p, int& s, Point* ch) {
    int i, k = 0;
    p1 = p2 = p[0];
    for (i = 1;i < n; i++) {
        if (p1.y - p[i].y > EPS || (zero(p1.y - p[i].y) && p1.x > p[i].x)) {
            p1 = p[k = i];
        }
        p2.x += p[i].x;
        p2.y += p[i].y;
    }
    p2.x /= n;
    p2.y /= n;
    p[k] = p[0];
    p[0] = p1;
```

```
    qsort(p + 1, n - 1, sizeof(Point), graham_cp);
    ch[0] = p[0];
    ch[1] = p[1];
    ch[2] = p[2];
    for (s = i = 3; i < n; ch[s++] = p[i++]) {
        for (; s > 2 && xmult(ch[s - 2], p[i], ch[s-1]) < -EPS; s--);
    }
}

//构造凸包接口函数，传入原始点集大小 n，点集 p(p 原有顺序被打乱!)
//返回凸包大小，凸包的点在 convex 中
//参数 maxsize 为 true 包含共线点，为 false 不包含共线点，缺省为 true
//参数 clockwise 为 true 顺时针构造，为 false 逆时针构造，缺省为 true
//在输入仅有若干共线点时算法不稳定，可能有此类情况请另行处理!
//不能去掉点集中重合的点
int graham(int n, Point * p, Point * convex, bool maxsize = true, bool dir = true)
{
    Point* temp = new Point[n];
    int s, i;
    _graham(n, p, s, temp);
    convex[0] = temp[0];
    n = 1;
    for (i = (dir ? 1 : (s - 1)); dir ? (i < s) : i ; i += (dir ? 1 : -1)) {
        if (maxsize || !zero(xmult(temp[i - 1], temp[i], temp[(i + 1) % s]))) {
            convex[n++] = temp[i];
        }
    }
    delete []temp;
    return n;
}


// CONVEX HULL II
// modified by mgmg 去掉点集中重合的点
#include <cstdlib>

const double EPS = 1e-8;

struct Point {
    double x, y;
};

inline bool zero(double x) {
    return ((x > 0 ? x : -x) < EPS);
}

//计算 cross product (P1 - P0)x(P2 - P0)
```

```
inline double xmult(const Point & p1, const Point & p2, const Point & p0) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}

//graham 算法顺时针构造包含所有共线点的凸包, O(nlogn)
Point p1, p2;
int graham_cp(const void * a, const void * b) {
    double ret = xmult(*((Point *)a), *((Point *)b), p1);
    return zero(ret) ? (xmult(*((Point *)a), *((Point *)b), p2) > 0 ? 1 : -1) : (re
t > 0 ? 1 : -1);
}
void _graham(int n, Point * p, int & s, Point * ch) {
    int i, k = 0;
    p1 = p2 = p[0];
    for (i = 1; i < n; i++) {
        if (p1.y - p[i].y > EPS || (zero(p1.y - p[i].y) && p1.x > p[i].x)) {
            p1 = p[k = i];
        }
        p2.x += p[i].x;
        p2.y += p[i].y;
    }
    p2.x /= n;
    p2.y /= n;
    p[k] = p[0];
    p[0] = p1;
    qsort(p + 1, n - 1, sizeof(Point), graham_cp);
    ch[0] = p[0];
    ch[1]=p[1];
    ch[2]=p[2];
    for (s = i = 3; i < n; ch[s++] = p[i++]) {
        for (;s > 2 && xmult(ch[s - 2], p[i], ch[s - 1]) < -EPS; s--);
    }
}

int wipesame_cp(const void * a, const void * b)
{
    if ((*(Point *)a).y < (*(Point *)b).y - EPS) {
        return -1;
    } else if ((*(Point *)a).y > (*(Point *)b).y + EPS) {
        return 1;
    } else if ((*(Point *)a).x < (*(Point *)b).x - EPS) {
        return -1;
    } else if ((*(Point *)a).x > (*(Point *)b).x + EPS) {
        return 1;
    } else {
        return 0;
    }
```

```
}

int _wipesame(Point * p, int n)
{
    int i, k;
    qsort(p, n, sizeof(Point), wipesame_cp);
    for (k = i = 1; i < n; i++) {
        if (wipesame_cp(p + i, p + i - 1) != 0) {
            p[k++] = p[i];
        }
    }
    return k;
}

//构造凸包接口函数，传入原始点集大小 n，点集 p(p 原有顺序被打乱!)
//返回凸包大小，凸包的点在 convex 中
//参数 maxsize 为 true 包含共线点，为 false 不包含共线点，缺省为 true
//参数 clockwise 为 true 顺时针构造，为 false 逆时针构造，缺省为 true
//在输入仅有若干共线点时算法不稳定，可能有此类情况请另行处理!
int graham(int n, Point * p, Point * convex, bool maxsize = true, bool dir = true)
{
    Point * temp = new Point[n];
    int s, i;
    n = _wipesame(p, n);
    _graham(n, p, s, temp);
    convex[0] = temp[0];
    n = 1;
    for (i = (dir ? 1 : (s - 1)); dir ? (i < s) : i; i += (dir ? 1 : -1)) {
        if (maxsize || !zero(xmult(temp[i - 1], temp[i], temp[(i + 1) % s]))) {
            convex[n++] = temp[i];
        }
    }
    delete [] temp;
    return n;
}


// CONVEX HULL III by VB

//构造凸包函数,传入原始点集大小 n,点集 p(p 原有顺序被打乱!)
//返回凸包大小,凸包的点在 ch 中
//参数 maxsize 为 true 包含共线点,为 false 不包含共线点,缺省为 false
//返回的凸包为逆时针顺序
//保留共线点时重点会引起算法不稳定，而所有点在一条直线上时中间点会重复出现
//不保留共线点时没有以上问题
#include <cstdlib>
```

```
const double EPS = 1e-8;

struct Point {
    double x, y;
};

inline bool zero(double x) {
    return ((x > 0 ? x : -x) < EPS);
}

//计算cross product (P1 - P0)x(P2 - P0)
inline double xmult(const Point & p1, const Point & p2, const Point & p0) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}

int cmp(const void * a, const void * b) {
    Point *p1 = (Point *) a, *p2 = (Point *)b;
    return zero(p1->y - p2->y) ? (p1->x > p2->x + EPS ? 1 : -1) : (p1->y > p2->y +
EPS ? 1 : -1);
}

int graham(int n, Point p[], Point ch[], bool maxsize = false) {
    const double e1 = maxsize ? EPS : -EPS;
    int i, j, k;
    if (n < 3) {
        for(i = 0; i < n; i++) {
            ch[i]=p[i];
        }
        return n;
    }
    qsort(p, n, sizeof(p[0]), cmp);
    ch[0] = p[0];
    ch[1] = p[1];
    for (i = j = 2; i < n; ch[j++] = p[i++]) {
        while (j > 1 && xmult(ch[j - 2], p[i], ch[j - 1]) > e1) {
            j--;
        }
    }
    ch[k = j++] = p[n - 2];
    for (i = n - 3; i > 0; ch[j++] = p[i--]) {
        while (j > k && xmult(ch[j - 2], p[i], ch[j - 1]) > e1) {
            j--;
        }
    }
    while (j > k && xmult(ch[j - 2], ch[0], ch[j - 1]) > e1) {
        j--;
    }
```

```
    return j;
}
```

网格

```
#include <cstdlib>

struct Point{
    int x, y;
};

int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}

//多边形上的网格点个数
int gridOnedge(int n, const Point * p) {
    int i, ret = 0;
    for (i = 0; i < n; i++) {
        ret += gcd(abs(p[i].x - p[(i + 1) % n].x), abs(p[i].y - p[(i + 1) % n].y));
    }
    return ret;
}

//多边形内的网格点个数
int gridInside(int n, const Point* p) {
    int i, ret = 0;
    for (i = 0; i < n; i++) {
        ret += p[(i + 1) % n].y * (p[i].x - p[(i + 2) % n].x);
    }
    return (abs(ret) - gridOnedge(n, p)) / 2 + 1;
}
```

圆

```
#include <cmath>

const double EPS = 1e-8;

struct Point {
    double x, y;
};

inline double xmult(const Point & p1, const Point & p2, const Point & p0) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}

inline double dis(const Point & p1, const Point & p2) {
    return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
```

```
}

double disptoline(const Point & p, const Point & l1, const Point & l2) {
    return fabs(xmult(p, l1, l2)) / dis(l1, l2);
}

Point intersection(const Point & u1, const Point & u2, const Point & v1, const Poin
t & v2) {
    Point ret = u1;
    double t = ((u1.x - v1.x) * (v1.y - v2.y) - (u1.y - v1.y) * (v1.x - v2.x)) / ((
u1.x - u2.x) * (v1.y - v2.y) - (u1.y - u2.y) * (v1.x - v2.x));
    ret.x += (u2.x - u1.x) * t;
    ret.y += (u2.y - u1.y) * t;
    return ret;
}

//判直线和圆相交，包括相切
int intersectLineCircle(const Point & c, double r, const Point & l1, const Point &
l2) {
    return disptoline(c, l1, l2) < r + EPS;
}

//判线段和圆相交，包括端点和相切
int intersectSegCircle(const Point & c, double r, const Point & l1, const Point & l
2) {
    double t1 = dis(c, l1) - r, t2 = dis(c, l2) - r;
    Point t = c;
    if (t1 < EPS || t2 < EPS) {
        return t1 > -EPS || t2 > -EPS;
    }
    t.x += l1.y - l2.y;
    t.y += l2.x - l1.x;
    return xmult(l1, c, t) * xmult(l2, c, t) < EPS && disptoline(c, l1, l2) - r < E
PS;
}

//判圆和圆相交，包括相切
int intersectCircleCircle(const Point & c1, double r1, const Point & c2, double r2)
 {
    return dis(c1, c2) < r1 + r2 + EPS && dis(c1, c2) > fabs(r1 - r2) - EPS;
}

//计算圆上到点p最近点，如p与圆心重合，返回p本身
Point dotToCircle(const Point & c, double r, const Point & p) {
    Point u, v;
    if (dis(p, c)<EPS) {
        return p;
```

```
    }
    u.x = c.x + r * fabs(c.x - p.x) / dis(c, p);
    u.y = c.y + r * fabs(c.y - p.y) / dis(c, p) * ((c.x - p.x) * (c.y - p.y) < 0 ?
-1 : 1);
    v.x = c.x - r * fabs(c.x - p.x) / dis(c, p);
    v.y = c.y - r * fabs(c.y - p.y) / dis(c, p) * ((c.x - p.x) * (c.y - p.y) < 0 ?
-1 : 1);
    return dis(u, p) < dis(v, p) ? u : v;
}

//计算直线与圆的交点，保证直线与圆有交点
//计算线段与圆的交点可用这个函数后判点是否在线段上
void intersectionLineCircle(const Point & c, double r, const Point & l1, const Poin
t & l2, Point & p1, Point & p2) {
    Point p = c;
    p.x += l1.y - l2.y;
    p.y += l2.x - l1.x;
    p = intersection(p, c, l1, l2);
    double t = sqrt(r * r - dis(p, c) * dis(p, c)) / dis(l1, l2);
    p1.x = p.x + (l2.x - l1.x) * t;
    p1.y = p.y + (l2.y - l1.y) * t;
    p2.x = p.x - (l2.x - l1.x) * t;
    p2.y = p.y - (l2.y - l1.y) * t;
}

//计算圆与圆的交点，保证圆与圆有交点，圆心不重合
void intersectionCircleCircle(const Point & c1, double r1, const Point & c2, double
 r2, Point & p1, Point & p2) {
    Point u, v;
    double t = (1 + (r1 * r1 - r2 * r2) / dis(c1, c2) / dis(c1, c2)) / 2;
    u.x = c1.x + (c2.x - c1.x) * t;
    u.y = c1.y + (c2.y - c1.y) * t;
    v.x = u.x + c1.y - c2.y;
    v.y = u.y - c1.x + c2.x;
    intersectionLineCircle(c1, r1, u, v, p1, p2);
}
```

整数函数

```
//整数几何函数库
//注意某些情况下整数运算会出界!
struct Point {
    int x, y;
};
struct Line {
    Point a, b;
};
```

```
inline int sign(int a) {
    return (a > 0 ? 1 : (a < 0 ? -1 : 0));
}

//计算cross product (P1-P0)x(P2-P0)
inline int xmult(const Point & p1, const Point & p2, const Point & p0) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}
inline int xmult(int x1, int y1, int x2, int y2, int x0, int y0) {
    return (x1 - x0) * (y2 - y0) - (x2 - x0) * (y1 - y0);
}

//计算dot product (P1-P0).(P2-P0)
inline int dmult(const Point & p1, const Point & p2, const Point & p0) {
    return (p1.x - p0.x) * (p2.x - p0.x) + (p1.y - p0.y) * (p2.y - p0.y);
}
inline int dmult(int x1, int y1, int x2, int y2, int x0, int y0) {
    return (x1 - x0) * (x2 - x0) + (y1 - y0) * (y2 - y0);
}

//判三点共线
bool dotsInline(const Point & p1, const Point & p2, const Point & p3) {
    return !xmult(p1, p2, p3);
}
bool dotsInline(int x1, int y1, int x2, int y2, int x3, int y3) {
    return !xmult(x1, y1, x2, y2, x3, y3);
}

//判点是否在线段上, 包括端点和部分重合
bool dotOnlineIn(const Point & p, const Line & l) {
    return !xmult(p, l.a, l.b) && (l.a.x - p.x) * (l.b.x - p.x) <= 0 && (l.a.y - p.
y) * (l.b.y - p.y) <= 0;
}
bool dotOnlineIn(const Point & p, const Point & l1, const Point & l2) {
    return !xmult(p, l1, l2) && (l1.x - p.x) * (l2.x - p.x) <= 0 && (l1.y - p.y) *
(l2.y - p.y) <= 0;
}
bool dotOnlineIn(int x, int y, int x1, int y1, int x2, int y2) {
    return !xmult(x, y, x1, y1, x2, y2) && (x1 - x) * (x2 - x) <= 0 && (y1 - y) * (
y2 - y) <= 0;
}

//判点是否在线段上, 不包括端点
bool dotOnlineEx(const Point & p, const Line & l) {
    return dotOnlineIn(p, l) && (p.x != l.a.x || p.y != l.a.y) && (p.x != l.b.x ||
p.y != l.b.y);
}
```

```
bool dotOnlineEx(const Point & p, const Point & l1, const Point & l2) {
    return dotOnlineIn(p, l1, l2) && (p.x != l1.x || p.y != l1.y) && (p.x != l2.x |
| p.y != l2.y);
}
bool dotOnlineEx(int x, int y, int x1, int y1, int x2, int y2) {
    return dotOnlineIn(x, y, x1, y1, x2, y2) && (x != x1 || y != y1) && (x != x2 ||
 y != y2);
}

//判两点在直线同侧，点在直线上返回 0
bool sameSide(const Point & p1, const Point & p2, const Line & l) {
    return sign(xmult(l.a, p1, l.b)) * xmult(l.a, p2, l.b) > 0;
}
bool sameSide(const Point & p1, const Point & p2, const Point & l1, const Point & l
2) {
    return sign(xmult(l1, p1, l2)) * xmult(l1, p2, l2) > 0;
}

//判两点在直线异侧，点在直线上返回 0
bool oppositeSide(const Point & p1, const Point & p2, const Line & l) {
    return sign(xmult(l.a, p1, l.b)) * xmult(l.a, p2, l.b) < 0;
}
bool oppositeSide(const Point & p1, const Point & p2, const Point & l1, const Point
 & l2) {
    return sign(xmult(l1, p1, l2)) * xmult(l1, p2, l2) < 0;
}

//判两直线平行
bool parallel(const Line & u, const Line & v) {
    return (u.a.x - u.b.x) * (v.a.y - v.b.y) ==(v.a.x - v.b.x) * (u.a.y - u.b.y);
}
bool parallel(const Point & u1, const Point & u2, const Point & v1, const Point & v
2) {
    return (u1.x - u2.x) * (v1.y - v2.y) == (v1.x - v2.x) * (u1.y - u2.y);
}

//判两直线垂直
bool perpendicular(const Line & u, const Line & v) {
    return (u.a.x - u.b.x) * (v.a.x - v.b.x) == -(u.a.y - u.b.y) * (v.a.y - v.b.y);
}
bool perpendicular(const Point & u1, const Point & u2, const Point & v1, const Poin
t & v2) {
    return (u1.x - u2.x) * (v1.x - v2.x) == -(u1.y - u2.y) * (v1.y - v2.y);
}

//判两线段相交，包括端点和部分重合
bool intersectIn(const Line & u, const Line & v) {
```

```
    if (!dotsInline(u.a, u.b, v.a) || !dotsInline(u.a, u.b, v.b)) {
        return !sameSide(u.a, u.b, v) && !sameSide(v.a, v.b, u);
    } else {
        return dotOnlineIn(u.a, v) || dotOnlineIn(u.b, v) || dotOnlineIn(v.a, u) ||
 dotOnlineIn(v.b, u);
    }
}
bool intersectIn(const Point & u1, const Point & u2, const Point & v1, const Point
& v2) {
    if (!dotsInline(u1, u2, v1) || !dotsInline(u1, u2, v2)) {
        return !sameSide(u1, u2, v1, v2) && !sameSide(v1, v2, u1, u2);
    } else {
        return dotOnlineIn(u1, v1, v2) || dotOnlineIn(u2, v1, v2) || dotOnlineIn(v1
, u1, u2) || dotOnlineIn(v2, u1, u2);
    }
}

//判两线段相交，不包括端点和部分重合
bool intersectEx(const Line & u, const Line & v) {
    return oppositeSide(u.a, u.b, v) && oppositeSide(v.a, v.b, u);
}
bool intersectEx(const Point & u1, const Point & u2, const Point & v1, const Point
& v2) {
    return oppositeSide(u1, u2, v1, v2) && oppositeSide(v1, v2, u1, u2);
}
```

组合

组合公式

- C(m,n)=C(m,m-n)
- C(m,n)=C(m-1,n)+C(m-1,n-1)
- derangement D(n) = n!(1 - 1/1! + 1/2! - 1/3! + ... + (-1)^n/n!)

  = (n-1)(D(n-2) + D(n-1))

  Q(n) = D(n) + D(n-1)

- Catalan numbers:

Ca(n)=C(2n-2,n-1)/n

- K-dimensional Catalan numbers:

  A(n) = 0! * 1! * .. * (k - 1)! * (k * n)! / (n! * (n + 1)! * .. * (n + k - 1)!)

- 2-d:

  1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900

- 3-d:

  1, 5, 42, 462, 6006, 87516, 1385670, 23371634, 414315330

- 4-d:

  1, 14, 462, 24024, 1662804, 140229804, 13672405890, 1489877926680

  求和公式,k = 1..n

- sum(k) = n(n+1)/2
- sum(2k-1) = n^2
- sum(k^2) = n(n+1)(2n+1)/6
- sum((2k-1)^2) = n(4n^2-1)/3
- sum(k^3) = (n(n+1)/2)^2
- sum((2k-1)^3) = n^2(2n^2-1)
- sum(k^4) = n(n+1)(2n+1)(3n^2+3n-1)/30
- sum(k^5) = n^2(n+1)^2(2n^2+2n-1)/12
- sum(k(k+1)) = n(n+1)(n+2)/3
- sum(k(k+1)(k+2)) = n(n+1)(n+2)(n+3)/4
- sum(k(k+1)(k+2)(k+3)) = n(n+1)(n+2)(n+3)(n+4)/5

排列组合生成

```
//genPerm 产生字典序排列 P(n, m)
//genComb 产生字典序组合 C(n, m)
//genPermSwap 产生相邻位对换全排列 P(n, n)
//产生元素用 1..n 表示
//dummy 为产生后调用的函数, 传入 a[]和 n, a[0]..a[n-1]为一次产生的结果
const int MAXN = 100;

void dummy(const int * a, int n) {
    //...
}
```

```
void genPermRc(int* a, int n, int m, int k, int* temp, bool * tag) {
    if (k == m) {
        dummy(temp, m);
    } else {
        for (int i = 0; i < n; i++) {
            if (!tag[i]) {
                temp[k] = a[i], tag[i] = true;
                genPermRc(a, n, m, k + 1, temp, tag);
                tag[i] = 0;
            }
        }
    }
}

void genPerm(int n, int m) {
    int a[MAXN], temp[MAXN];
    bool tag[MAXN] = {false};
    for (int i = 0; i < n; i++) {
        a[i] = i + 1;
    }
    genPermRc(a, n, m, 0, temp, tag);
}

void genCombRc(int * a, int s, int e, int m, int & count, int * temp) {
    if (m == 0) {
        dummy(temp, count);
    } else {
        for (int i = s; i <= e - m + 1; i++) {
            temp[count++] = a[i];
            genCombRc(a, i + 1, e, m - 1, count, temp);
            count--;
        }
    }
}

void genComb(int n, int m) {
    int a[MAXN], temp[MAXN], count = 0;
    for (int i = 0; i < n; i++) {
        a[i] = i + 1;
    }
    genCombRc(a, 0, n-1, m, count, temp);
}

void genPermSwapRc(int* a, int n, int k, int* pos, int * dir) {
    int p1, p2, t;
    if (k == n) {
        dummy(a, n);
```

```
    } else {
        genPermSwapRc(a, n, k + 1, pos, dir);
        for (int i = 0; i < k; i++) {
            p2 = (p1 = pos[k]) + dir[k];
            t = a[p1];
            a[p1] = a[p2];
            a[p2] = t;
            pos[a[p1] - 1] = p1;
            pos[a[p2] - 1] = p2;
            genPermSwapRc(a, n, k + 1, pos, dir);
        }
        dir[k] = -dir[k];
    }
}

void genPermSwap(int n) {
    int a[MAXN], pos[MAXN], dir[MAXN];
    for (int i = 0; i < n; i++) {
        a[i] = i + 1;
        pos[i] = i;
        dir[i] = -1;
    }
    genPermSwapRc(a, n, 0, pos, dir);
}
```

生成 gray 码

```
//求置换的循环节, polya 原理
//perm[0..n-1]为 0..n-1 的一个置换(排列)
//返回置换最小周期, num 返回循环节个数
const int MAXN = 1000;

int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}

int polya(int * perm, int n, int & num) {
    int i, j, p, v[MAXN] = {0}, ret = 1;
    for (num = i = 0;i < n; i++) {
        if (!v[i]) {
            num++;
            p = i;
            for (j=0; !v[p = perm[p]]; j++) {
                v[p] = 1;
            }
            ret *= j / gcd(ret, j);
        }
    }
```

```
    return ret;
}
//生成 reflected gray code
//每次调用 gray 取得下一个码
//000...000 是第一个码,100...000 是最后一个码
void gray(int n, int *code) {
    int t = 0;
    for (int i = 0; i < n; t += code[i++]);
    if (t & 1) {
        for (n--; !code[n]; n--);
    }
    code[n - 1] = 1 - code[n - 1];
}
```

置换(polya)

```
//求置换的循环节, polya 原理
//perm[0..n-1]为 0..n-1 的一个置换(排列)
//返回置换最小周期, num 返回循环节个数
const int MAXN = 1000;

int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}

int polya(int * perm, int n, int & num) {
    int i, j, p, v[MAXN] = {0}, ret = 1;
    for (num = i = 0;i < n; i++) {
        if (!v[i]) {
            num++;
            p = i;
            for (j=0; !v[p = perm[p]]; j++) {
                v[p] = 1;
            }
            ret *= j / gcd(ret, j);
        }
    }
    return ret;
}
```

字典序全排列

```
//字典序全排列与序号的转换
int perm2Num(int n, const int * p) {
    int ret = 0, k = 1;
    for (int i = n - 2; i >= 0; k *= n - (i--)) {
        for (int j = i + 1; j < n; j++) {
            if (p[j] < p[i]) {
                ret += k;
```

```
            }
        }
    }
    return ret;
}

void num2Perm(int n, int * p, int t) {
    for (int i = n - 1; i >= 0; i--) {
        p[i] = t % (n - i);
        t /= n - i;
    }
    for (int i = n - 1; i; i--) {
        for (int j = i - 1; j >= 0; j--) {
            if (p[j] <= p[i]) {
                p[i]++;
            }
        }
    }
}
```

字典序组合

```
//字典序组合与序号的转换
//comb 为组合数 C(n, m), 必要时换成大数, 注意处理 C(n, m) = 0 | n < m
int comb(int n, int m) {
    int ret = 1;
    m = m < (n - m) ? m :(n - m);
    for (int i = n - m + 1; i <= n; ret *= (i++));
    for (int i = 1; i <= m; ret /= (i++));
    return m < 0 ? 0 : ret;
}

int comb2Num(int n, int m, const int * c) {
    int ret = comb(n, m);
    for (int i = 0; i < m; i++) {
        ret -= comb(n - c[i], m - i);
    }
    return ret;
}

void num2Comb(int n, int m, int * c, int t) {
    int j = 1, k;
    for (int i = 0; i < m; c[i++] = j++) {
        for (; t > (k = comb(n - j, m - i - 1)); t -= k, j++);
    }
}
```

结构

Splay

```
#include<stdio.h>
#include<iostream>

using namespace std;

struct snode;

typedef struct snode * stree;
typedef struct snode * Posi;
typedef int eleType;

const int MAXDEPTH = 256;//最大树高

struct snode{
    stree son[2];
    eleType val;
    /*int num;*///记录以此节点为根的树有多少个节点，需要使用 num 域时请去掉程序中所有/**/注释
    //bool rev;//记录以此节点为根的树是否被翻转，需要使用 rev 域时请去掉程序中所有//注释
};

static Posi null = NULL;

void init(){//请首先调用此函数
    null = new snode;
    memset(null,0,sizeof(snode));
    null->son[0] = null->son[1] = null;
}

inline void check(Posi X){//检查并维护节点 X 的翻转状态
    /*if(X->rev){
        X->rev = false;
        swap(X->son[0],X->son[1]);
        if(X->son[0] != null)
            X->son[0]->rev ^= true;
        if(X->son[1] != null)
            X->son[1]->rev ^= true;
    }*/
}

void print(Posi X){//打印树的每一个节点
    if(X != null){
        check(X);
```

```
        print(X->son[0]);
        printf("%d ",X->val);
        print(X->son[1]);
    }
}

/*void show(Posi X){//打印树的每个子树的节点个数
    if(X != null){
        check(X);
        show(X->son[0]);
        if(first) first = false;
        else printf(" ");
        printf("%d",X->num);
        show(X->son[1]);
    }
}*/

inline Posi rotate(Posi X, bool s){
    check(X);
    Posi tmp = X->son[s];
    /*X->num -= tmp->num;*/
    if(tmp != null){
        /*tmp->num += X->num;
        X->num += tmp->son[!s]->num;*/
        check(tmp);
        X->son[s] = tmp->son[!s];
        tmp->son[!s] = X;
        return tmp;
    }
    return X;
}

void update_num(Posi X){
    /*int pos;
    Posi q[MAXDEPTH], cur;

    check(X);
    for(int i = 0;i < 2;i++){
        pos = 0;
        cur = X->son[1 - i];
        while(cur != null){
            q[pos++] = cur;
            cur = cur->son[i];
            check(cur);
        }
        for(int j = pos - 1;j >= 0;j--){
            q[j]->num = q[j]->son[0]->num + q[j]->son[1]->num + 1;
```

```
        }
    }
    X->num = X->son[0]->num + X->son[1]->num + 1;*/
}

//将最接近item的节点旋转至树根
stree splay(eleType item, Posi X){
    static snode h;
    Posi m[2];
    bool s;

    if(X == null) return null;
    h.son[0] = h.son[1] = null;m[0] = m[1] = &h;
    null->val = item;
    check(X);
    while(item != X->val){
        s = (item > X->val);
        if(!s) if(item < X->son[s]->val)
            X = rotate(X,s);
        else{} else if(item > X->son[s]->val)
            X = rotate(X,s);
        if(X->son[s] == null) break;
        m[!s]->son[s] = X;m[!s] = X;
        /*X->num -= X->son[s]->num;*/
        X = X->son[s];
        m[!s]->son[s] = null;
        check(X);
    }
    m[0]->son[1] = X->son[0];
    m[1]->son[0] = X->son[1];
    X->son[0] = h.son[1];
    X->son[1] = h.son[0];
    update_num(X);
    return X;
}

//将第n(1-TotalNumber)个节点旋转至树根
/*stree splayn(int n, Posi X){
    static snode h;
    Posi m[2];
    bool s;

    if(X == null) return null;
    if(n > X->num) n = X->num;if(n < 1) n = 1;
    h.son[0] = h.son[1] = null;m[0] = m[1] = &h;
    check(X);
    while((s = (n <= X->son[0]->num)) || (X->num < X->son[1]->num + n)){
```

```
        s ^= true;
        if(!s) if(n < X->son[0]->son[0]->num)
            X = rotate(X,0);
        else{} else if(X->num < X->son[1]->son[1]->num + n){
            X = rotate(X,1);
        }
        if(X->son[s] == null) break;
        if(s) n -= (X->num - X->son[1]->num);m[!s]->son[s] = X;m[!s] = X;
        X->num -= X->son[s]->num;X = X->son[s];m[!s]->son[s] = null;
        check(X);
    }
    m[0]->son[1] = X->son[0];m[1]->son[0] = X->son[1];
    X->son[0] = h.son[1];X->son[1] = h.son[0];
    update_num(X);
    return X;
}*/

stree insert(eleType item, stree T){
    static Posi node = NULL;
    bool s;
    if(node == NULL)
        node = new snode;
    node->val = item;//node->rev = false;
    if(T == null){
        node->son[0] = node->son[1] = null;
        /*node->num = 1;*/
        T = node;
    }else{
        T = splay(item, T);
        if(item == T->val){
            /*(T->num)++;*/
            return T;
        }
        s = (T->val < item);
        /*T->num -= T->son[s]->num;*/
        node->son[s] = T->son[s];
        node->son[!s] = T;
        /*node->num = node->son[s]->num + node->son[!s]->num + 1;*/
        T->son[s] = null;
        T = node;
    }
    node = NULL;
    return T;
}

stree remove(eleType item, stree T){
    Posi root;
```

```
    if(T != null){
        T = splay(item, T);
        if(item == T->val){
            if(T->son[0] == null)
                root = T->son[1];
            else{
                root = T->son[0];
                root = splay(item, root);
                root->son[1] = T->son[1];
                root->num += root->son[1]->num;
            }
            delete T;
            return root;
        }
    }
    return T;
}
```

Trie 图

```
/*
    Trie 图，即用于多串匹配的字符串自动机。
    对于字符串 S 中是否存在关键字 s_1,s_2,..,s_n 的匹配的问题，
    可以用 O((|s_1|+|s_2|+...+|s_n|)|∑|)的时间预处理，O(|S|)的时间回答询问。
    若深入理解，也可根据具体情况扩展之，以解决很多字符串有关的题。
    注意事项：
        1. 字符的类型、字符集的大小可改。
        2. ELEMENT_MAX 一般可设为|s_1|+|s_2|+...+|s_n|的最大可能值。
        3. 传递的字符串中每个字符应在[0,SIGMA]的区间内。
    使用方法：
        1. 处理每个测试点前调用 TrieGraph::init()以初始化。
        2. 用 insert 插入 s_1,s_2,...
        3. 调用 build_graph 建立 trie 图。
        4. 调用 match 查询 S 中是否存在匹配。
*/

#include <cstdio>
#include <cstring>
using namespace std;

typedef char char_t;//字符的类型，需要的话可以改成 int 等
const int SIGMA=26;//字符集的大小
const int ELEMENT_MAX=5000;//trie 图中最多可能的节点数，用于内存管理

struct TrieGraph{
    struct trie{
        bool match;
        trie *pre,*child[SIGMA];
```

```
        trie():match(false),pre(0){memset(child,0,sizeof(child));}
        void* operator new(size_t, void *p){return p;}
    }root,superroot;
    static char storage[ELEMENT_MAX*sizeof(trie)];
    static trie* data;
    static void init(){data=(trie*)storage;}
    void insert(char_t* s,int n){//在 trie 中插入一个长度为 n 的字符串 s
        //如果需要在 trie 节点中记录某些信息，一般是在这个函数里添加
        trie* t=&root;
        for(int i=0;i<n;++i){
            char_t c=s[i];
            if(!t->child[c])
                t->child[c]=new((void*)data++) trie;
            t=t->child[c];
        }
        t->match=true;
    }
    void build_graph(){//所有的插入完毕后，将 trie 树扩充为图
        static trie* Q[ELEMENT_MAX];
        superroot.pre=root.pre=&superroot;
        for(int i=0;i<SIGMA;++i)
            superroot.child[i]=&root;
        int head=0,tail=0;
        Q[tail++]=&root;
        while(head!=tail){
            trie* t=Q[head++];
            for(int i=0;i<SIGMA;++i){
                if(t->child[i]){
                    t->child[i]->pre=t->pre->child[i];
                    Q[tail++]=t->child[i];
                }
                else
                    t->child[i]=t->pre->child[i];
            }
        }
    }
    bool match(char_t* s,int n){//返回长度为 n 的字符串 s 中有否已被插入的字符串的匹配
        trie* t=&root;
        for(int i=0;i<n;++i)
            if((t=t->child[s[i]])->match)
                return true;
        return false;
    }
};
char TrieGraph::storage[ELEMENT_MAX*sizeof(trie)];
TrieGraph::trie* TrieGraph::data;
```

```
/*
    示例程序：
        输入 n 和 m，以及 n 个字符串 s_1..S_n;
        以下 m 次询问，每次询问一个字符串是否与 s_1..s_n 中的某一个匹配，输出 Yes 或 No。
*/

int main(){
    int N,M;
    char s[256];
    while(2==scanf("%d%d",&N,&M)){
        gets(s);
        TrieGraph::init();
        TrieGraph g;
        for(int i=0;i<N;++i){
            gets(s);
            int n=strlen(s);
            for(int i=0;i<n;++i)
                s[i]-='a';
            g.insert(s,n);
        }
        g.build_graph();
        for(int i=0;i<N;++i){
            gets(s);
            int n=strlen(s);
            for(int i=0;i<n;++i)
                s[i]-='a';
            puts(g.match(s,n)?"Yes":"No");
        }
    }
}
```

并查集

```
//带路径压缩的并查集,用于动态维护查询等价类
//图论算法中动态判点集连通常用
//维护和查询复杂度略大于O(1)
//集合元素取值1..MAXN-1(注意0不能用!),默认不等价
#include <cstring>

const int MAXN = 100000;
#define _run(x) for(; p[t = x]; x = p[x], p[t] = (p[x] ? p[x] : x))
#define _run_both _run(i); _run(j)

class DSet {
public:
    int p[MAXN],t;

    void init() {
```

```
        memset(p, 0, sizeof(p));
    }

    void setFriend(int i, int j) {
        _run_both;
        p[i] = (i == j ? 0 : j);
    }

    bool isFriend(int i, int j) {
        _run_both;
        return i == j && i;
    }
};

//带路径压缩的并查集扩展形式
//用于动态维护查询 friend-enemy 型等价类
//维护和查询复杂度略大于 O(1)
//集合元素取值 1..MAXN-1(注意 0 不能用!),默认无关
#include <cstring>
#include <cstdlib>

const int MAXN = 100000;

#define sig(x) ((x)>0?1:-1)
#define _run(x) for(; p[t = abs(x)]; x = sig(x) * p[abs(x)], p[t] = sig(p[t]) * (p[
abs(x)] ? p[abs(x)] : abs(p[t])))
#define _run_both _run(i); _run(j)
#define _set_side(x) p[abs(i)] = sig(i) * (abs(i) == abs(j) ? 0 : (x) * j)
#define _judge_side(x) (i == (x) * j && i)

class DSet {
public:
    int p[MAXN], t;

    void init() {
        memset(p, 0, sizeof(p));
    }

    bool setFriend(int i, int j) {
        _run_both;
        _set_side(1);
        return !_judge_side(-1);
    }

    bool setEnemy(int i, int j) {
        _run_both;
        _set_side(-1);
```

```
        return !_judge_side(1);
    }

    bool isFriend(int i, int j) {
        _run_both;
        return _judge_side(1);
    }

    bool isEnemy(int i, int j) {
        _run_both;
        return _judge_side(-1);
    }
};
```

子段和

```
//求sum{[0..n-1]}
//维护和查询复杂度均为O(logn)
//用于动态求子段和,数组内容保存在Sum.a[]中
//可以改成其他数据类型
#include <cstring>

const int MAXN = 10000;

inline int lowbit(int x) {
    return (x & (x ^ (x - 1)));
}

template<class elemType>
class Sum{
public:
    elemType a[MAXN], c[MAXN], ret;
    int n;

    void init(int i) {
        memset(a, 0, sizeof(a));
        memset(c, 0, sizeof(c));
        n = i;
    }

    void update(int i, elemType v) {
        v -= a[i];
        a[i] += v;
        for (i++; i <= n; i += lowbit(i)) {
            c[i - 1] += v;
        }
    }
```

```
    elemType query(int i) {
        for (ret = 0; i; i ^= lowbit(i)) {
            ret += c[i-1];
        }
        return ret;
    }
};
```

子阵和

```
//求sum{a[0..m-1][0..n-1]}
//维护和查询复杂度均为O(logm*logn)
//用于动态求子阵和，数组内容保存在Sum.a[][]中
//可以改成其他数据类型
#include <cstring>

const int MAXN = 10000;

inline int lowbit(int x) {
    return (x & (x ^ (x - 1)));
}


template<class elemType>
class Sum{
    elemType a[MAXN][MAXN], c[MAXN][MAXN], ret;
    int m, n, t;
    void init(int i, int j) {
        memset(a, 0, sizeof(a));
        memset(c, 0, sizeof(c));
        m=i;
        n=j;
    }

    void update(int i, int j, elemType v) {
        for (v -= a[i][j], a[i++][j++] += v, t = j; i <= m; i += lowbit(i)) {
            for (j = t; j <= n; j += lowbit(j)) {
                c[i - 1][j - 1] += v;
            }
        }
    }

    elemType query(int i, int j) {
        for (ret = 0, t = j; i; i ^= lowbit(i)) {
            for (j = t; j; j ^= lowbit(j)) {
                ret += c[i - 1][j - 1];
            }
        }
```

```
        return ret;
    }
};
```

左偏树

```
// 左偏树是可以高效做合并操作的堆

#include <algorithm>    // swap
#include <functional>    // less

template<typename T = int, typename Pred = less<T> >
struct LeftistTree
{
    struct node_type
    {
        T v;
        int d;
        node_type *l, r;
        node_type(T v, int d)
        {
            this->v = v;
            this->d = d;
            l = NULL;
            r = NULL;
        }
        ~node_type()
        {
            delete l;
            delete r;
        }
    };
private:
    node_type root;
    // 比较
    static Pred pr;
    // 核心操作，将以 l 和 r 为根的左偏树合并，返回新的根节点，复杂度 O(lgn)
    static node_type* merge(node_type* l, node_type* r)
    {
        if (l == NULL) {
            return r;
        }
        if (r == NULL) {
            return l;
        }

        if (pr(r->v, l->v)) {
            swap(l, r);
```

```
        }
        l->r = merge(l->r, r);
        if (l->r != NULL && (l->l == NULL || l->r->d > l->l->d)) {
            swap(l->l, l->r);
        }
        if (l->r == NULL) {
            l->d = 0;
        }
        else {
            l->d = l->r->d + 1;
        }

        return l;
    }
public:
    LeftistTree()
    {
        root = NULL;
    }
    ~LeftistTree()
    {
        delete root;
    }
    // 合并操作将让被合并的树变为空
    void merge(LeftistTree& t)
    {
        root = merge(root, t.root);
        t.root = NULL;
    }
    void push(T v)
    {
        root = merge(root, new node_type(v, 0));
    }
    void pop()
    {
        node_type *l = root->l, *r = root->r;

        root->l = NULL;
        root->r = NULL;
        delete root;
        root = merge(l, r);
    }
    T front()
    {
        return root->v;
    }
};
```

区段最小值查询（RMQ）

```
// RMQ
// MAXL = ceil(lg(MAXN))
// 根据需要重写以下函数

#define BIN(i) (1 << (i))
#define HLF(i) (BIN(i) >> 1)
#define PRE(i) ((i) > 0 ? (i) - 1 : 0)

// 为了省时间可以把 lg(x) 做成表，可以参考位运算加速
// 注意 lg(1 << i) = i - 1; lg((1 << i) + 1) = i;
const double eps = 1e-8;
const double ln2 = log(2.0);
inline int lg2(double x) { return (int)floor(fabs(log(x) / ln2 - eps)); }

/************ 一维 ************/

// MAXL = min{(1 << MAXL) >= MAXN};

template<int MAXL, class T = int, int MAXN = 1 << MAXL>
struct RMQ
{
    T e[MAXN];
    int rmq[MAXL][MAXN];

    // 重写 cmp，比较两个下标，返回较"小"下标
    int cmp(int l, int r) { return e[l] <= e[r] ? l : r; }

    // 请直接对 e 赋值后调用
    void init(const int n) {
        for (int i = 0; i < n; i++)
            rmq[0][i] = i;
        for (int i = 0; BIN(i + 1) < n; i++)
            for (int j = 0; j <= n - BIN(i + 1); j++)
                rmq[i + 1][j] = cmp(rmq[i][j], rmq[i][j + BIN(i)]);
    }

    // [l, r) (l < r)
    int index(int l, int r) { int b = lg2(r - l); return cmp(rmq[b][l], rmq[b][r -
(1 << b)]); }
    T value(int l, int r) { return e[index(l, r)]; }
};


/************ 二维 ************/
```

```
// 如果MLE就把int改成short
typedef pair<int, int> IndexType;

// MAXR = min{(1 << MAXR) >= MAXM};
// MAXC = min((1 << MAXC) >= MAXN);

template<int MAXR, int MAXC, class T = int, int MAXM = 1 << MAXR, int MAXN = 1 << M
AXC>
struct RMQ2
{
    T e[MAXM][MAXN];
    IndexType rmq[MAXR][MAXC][MAXM][MAXN];

    IndexType cmp(const IndexType& lhs, const IndexType& rhs) {
        return e[lhs.first][lhs.second] <= e[rhs.first][rhs.second] ? lhs : rhs;
    }

    void init(int m, int n) {
        for (int x = 0; x < m; x++)
            for (int y = 0; y < n; y++)
                rmq[0][0][x][y] = make_pair(x, y);
        for (int i = 0, ii; ii = PRE(i), BIN(i) < m; i++)
            for (int j = 0, jj; jj = PRE(j), BIN(j) < n; j++)
                for (int x = 0, xx; xx = HLF(i), x <= m - BIN(i); x++)
                    for (int y = 0, yy; yy = HLF(j), y <= n - BIN(j); y++)
                        rmq[i][j][x][y] = cmp(
                                cmp(rmq[ii][jj][x]     [y], rmq[ii][jj][x]     [y +
 yy]),
                                cmp(rmq[ii][jj][x + xx][y], rmq[ii][jj][x + xx][y +
 yy])
                                );
    }

    IndexType index(int x1, int y1, int x2, int y2) {
        int xx = lg2(x2 - x1), yy = lg2(y2 - y1);
        return cmp(
                cmp(rmq[xx][yy][x1]            [y1], rmq[xx][yy][x1]            [y2
 - (1 << yy)]),
                cmp(rmq[xx][yy][x2 - (1 << xx)][y1], rmq[xx][yy][x2 - (1 << xx)][y2
 - (1 << yy)])
                );
    }

    T value(int x1, int y1, int x2, int y2){
        IndexType i = index(x1, y1, x2, y2);
        return e[i.first][i.second];
    }
```

```
};

/*
可以在开头定义一个全局变量，代替用浮点函数的 lg2

template<int MAXN>
struct LG2
{
    int lg2[MAXN + 1];

    LG2()
    {
        lg2[0] = -1;
        for (int i = 1; i <= MAXN; i++) {
            lg2[i] = lg2[i - 1] + ((i & (i - 1)) == 0);
        }
    }

    int operator()(int x) const { return lg2[x]; }
};

LG2<65536> lg2;
 */
```

数论

整除规则

最后 n 位可以被 2^n 整除，则原数被 2^n 整除
各位数和可以被 3,9 整除，则原数被 3,9 整除
最后 n 位可以被 5^n 整除，则原数被 5^n 整除

对于其他的小素数有通用的方法：

删除最低位(设为 d)，剩下的数减去 y*d 得到的新数被 x 整除，则原数可以被 x 整除.(此过程可以重复直到数小到可以直接判)

x	y
7	2
11	1
13	9
17	5
19	17
23	16
29	26
31	3
37	11
41	4
43	30
47	14

阶乘最后非 0 位

```
//求阶乘最后非零位,复杂度 O(nlogn)
//返回该位,n 以字符串方式传入
#include <cstring>
const int MAXN = 10000;

int lastdigit(char * buf) {
    const int mod[20] =  {
        1, 1, 2, 6, 4, 2, 2, 4, 2, 8, 4, 4, 8, 4, 6, 8, 8, 6, 8, 2
    };
    int len = strlen(buf), a[MAXN], i, c, ret = 1;
    if (len == 1) {
        return mod[buf[0] - '0'];
    }
    for (i = 0; i < len; i++) {
        a[i] = buf[len - 1 - i] - '0';
    }
    for (; len; len -= !a[len - 1]) {
        ret = ret * mod[a[1] % 2 * 10 + a[0]] % 5;
        for (c = 0, i = len - 1; i >= 0; i--) {
            c = c * 10 + a[i];
```

```
            a[i] = c / 5;
            c %= 5;
        }
    }
    return ret + ret % 2 * 5;
}
```

模线性方程组

```
//扩展 Euclid 求解 gcd(a,b)=ax+by
int extGcd(int a, int b, int & x, int & y) {
    int t, ret;
    if (!b) {
        x = 1;
        y = 0;
        return a;
    }
    ret = extGcd(b, a % b, x, y);
    t = x;
    x = y;
    y = t - a / b * y;
    return ret;
}

//计算 m^a, O(loga), 本身没什么用, 注意这个按位处理的方法 :-P
int exponent(int m, int a) {
    int ret = 1;
    for (; a; a >>= 1, m *= m) {
        if (a & 1) {
            ret *= m;
        }
    }
    return ret;
}

//计算幂取模 a^b mod n, O(logb)
int modularExponent(int a, int b, int n) {
    //a^b mod n
    int ret = 1;
    for (; b; b >>= 1, a = (int) ((long long) a * a % n)) {
        if (b & 1) {
            ret = (int) ((long long) ret * a % n);
        }
    }
    return ret;
}

//求解模线性方程 ax=b (mod n)
```

```
//返回解的个数,解保存在 sol[]中
//要求 n>0,解的范围 0..n-1
int modularLinear(int a, int b, int n, int * sol) {
    int d, e, x, y, i;
    d = extGcd(a, n, x, y);
    if (b % d) {
        return 0;
    }
    e = (x *(b / d) % n + n) % n;
    for (i = 0; i < d; i++) {
        sol[i] = (e + i *(n / d)) % n;
    }
    return d;
}

//求解模线性方程组(中国余数定理)
//  x = b[0] (mod w[0])
//  x = b[1] (mod w[1])
//  ...
//  x = b[k-1] (mod w[k-1])
//要求 w[i]>0,w[i]与 w[j]互质,解的范围 1..n,n=w[0]*w[1]*...*w[k-1]
int modularLinearSystem(int b[], int w[], int k) {
    int d, x, y, a = 0, m, n = 1, i;
    for (i = 0; i < k; i++) {
        n *= w[i];
    }
    for (i = 0; i < k; i++) {
        m = n / w[i];
        d = extGcd(w[i], m, x, y);
        a = (a + y * m * b[i]) % n;
    }
    return (a + n) % n;
}
```

素数

```
//用素数表判定素数,先调用 initPrime
int plist[10000], pcount = 0;

bool prime(int n) {
    int i;
    if ((n != 2 && !(n % 2)) || (n != 3 && !(n % 3)) || (n != 5 && !(n % 5)) || (n
!= 7 && !(n % 7))) {
        return false;
    }
    for (i = 0; plist[i] * plist[i] <= n; i++) {
        if (!(n % plist[i])) {
            return false;
```

```
        }
    }
    return n > 1;
}

void initPrime() {
    plist[pcount++] = 2;
    for (int i = 3; i < 50000; i += 2) {
        if (prime(i)) {
            plist[pcount++] = i;
        }
    }
}

//miller rabin
//判断自然数 n 是否为素数
//time 越高失败概率越低,一般取 10 到 50
#include <cstdlib>

//a^b mod n
int modularExponent(int a, int b, int n) {
    int ret = 1;
    for (; b; b >>= 1, a = (int) ((long long) a * a % n)) {
        if (b & 1) {
            ret = (int) ((long long) ret * a % n);
        }
    }
    return ret;
}

// Carmicheal number: 561, 41041, 825265, 321197185,
// 5394826801, 232250619601, 9746347772161, 1436697831295441, 60977817398996785,
// 7156857700403137441, 1791562810662585767521, 8767496993623482137760 1
bool millerRabin(int n, int time = 10) {
    if (n == 1 || (n != 2 && !(n % 2)) || (n != 3 && !(n % 3)) || (n != 5 && !(n %
5)) || (n != 7 && !(n % 7))) {
        return false;
    }
    while (time--) {
        if (modularExponent(((rand() & 0x7fff << 16) + rand() & 0x7fff + rand() & 0
x7fff) % (n - 1) + 1, n - 1, n) != 1) {
            return false;
        }
    }
    return true;
}
```

欧拉函数

```
int gcd(int a, int b) {
    return b ? gcd(b, a % b): a;
}

inline int lcm(int a, int b) {
    return a / gcd(a, b) * b;
}

//求 1..n-1 中与 n 互质的数的个数
int eular(int n) {
    int ret = 1, i;
    for (i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            n /= i;
            ret *= i - 1;
            while (n % i == 0) {
                n /= i;
                ret *= i;
            }
        }
    }
    if (n > 1) {
        ret *= n - 1;
    }
    return ret;
}
```

分解质因数

```
//分解质因数
//primeFactor()传入 n, 返回不同质因数的个数
//f 存放质因数，nf 存放对应质因数的个数
//先调用 initPrime()，其中第二个 initPrime()更快

#include <cmath>

const int PSIZE = 100000;

int plist[PSIZE], pcount = 0;

bool prime(int n) {
    if ((n != 2 && !(n % 2)) || (n != 3 && !(n % 3)) || (n != 5 && !(n % 5)) || (n
!= 7 && !(n % 7))) {
        return false;
    }
    for (int i = 0; plist[i] * plist[i] <= n; ++i) {
```

```
        if (n % plist[i] == 0) {
            return false;
        }
    }
    return n > 1;
}

void initPrime() {
    plist[pcount++] = 2;
    for (int i = 3; i < 100000; i += 2) {
        if (prime(i)) {
            plist[pcount++] = i;
        }
    }
}

int primeFactor(int n, int * f, int * nf) {
    int cnt = 0;
    int n2 = (int) sqrt((double) n);
    for (int i = 0; n > 1 && plist[i] <= n2; ++i) {
        if (n % plist[i] == 0) {
            for (nf[cnt] = 0; n % plist[i] == 0; n /= plist[i]) {
                ++nf[cnt];
            }
            f[cnt++] = plist[i];
        }
    }
    if (n > 1) {
        nf[cnt] = 1;
        f[cnt++] = n;
    }
    return cnt;
}

//产生 MAXN 以内的所有素数
//note:2863311530 就是 10101010101010101010101010101010
//给所有 2 的倍数赋初值
#include <cmath>

const int MAXN = 100000000;
unsigned int plist[6000000], pcount;
unsigned int isPrime[(MAXN >> 5) + 1];

#define setbitzero(a) (isPrime[(a) >> 5] &= (~(1 << ((a) & 31))))
#define setbitone(a) (isPrime[(a) >> 5] |= (1 << ((a) & 31)))
#define ISPRIME(a) (isPrime[(a) >> 5] & (1 << ((a) & 31)))
```

```
void initPrime() {
    int i, j, m;
    int t = (MAXN >> 5) + 1;
    for (i = 0; i < t; ++i) {
        isPrime[i] = 2863311530;
    }
    plist[0] = 2;
    setbitone(2);
    setbitzero(1);
    m = (int) sqrt((double) MAXN);
    pcount = 1;
    for (i = 3; i <= m; i += 2) {
        if (ISPRIME(i)) {
            plist[pcount++] = i;
            for (j = i << 1; j <= MAXN; j += i) {
                setbitzero(j);
            }
        }
    }
    if (!(i & 1)) {
        ++i;
    }
    for (; i <= MAXN; i += 2) {
        if (ISPRIME(i)) {
            plist[pcount++] = i;
        }
    }
}
```

数值计算

定积分计算(Romberg)

```
/*  Romberg 求定积分
    输入：积分区间[a,b]，被积函数 f(x,y,z)
    输出：积分结果

    f(x,y,z)示例：
    double f0( double x, double l, double t )
    {
        return sqrt(1.0+l*l*t*t*cos(t*x)*cos(t*x));
    }
*/
#include <cmath>

double romberg(double a, double b, double(*f)(double x, double y, double z), double
 eps, double l, double t) {
    const int MAXN = 1000;
    int i, j, temp2, min;
    double h, R[2][MAXN], temp4;

    for (i = 0; i < MAXN; i++) {
        R[0][i] = 0.0;
        R[1][i] = 0.0;
    }
    h = b - a;
    min = (int)(log(h * 10.0) / log(2.0)); //h should be at most 0.1
    R[0][0] = ((*f)(a, l, t) + (*f)(b, l, t)) * h * 0.50;
    i = 1;
    temp2 = 1;
    while (i < MAXN) {
        i++;
        R[1][0] = 0.0;
        for (j = 1; j <= temp2; j++) {
            R[1][0] += (*f)(a + h *((double)j - 0.50), l, t);
        }
        R[1][0] = (R[0][0] + h * R[1][0]) * 0.50;
        temp4 = 4.0;
        for (j = 1; j < i; j++) {
            R[1][j] = R[1][j - 1] + (R[1][j - 1] - R[0][j - 1]) / (temp4 - 1.0);
            temp4 *= 4.0;
        }
        if ((fabs(R[1][i - 1] - R[0][i - 2]) < eps) && (i > min)) {
            return R[1][i - 1];
        }
```

```
        h *= 0.50;
        temp2 *= 2;
        for (j = 0; j < i; j++) {
            R[0][j] = R[1][j];
        }
    }
    return R[1][MAXN - 1];
}

double integral(double a, double b, double(*f)(double x, double y, double z), doubl
e eps, double l, double t) {
    const double PI = 3.1415926535897932;
    int n;
    double R, p, res;

    n = (int)(floor)(b * t * 0.50 / PI);
    p = 2.0 * PI / t;
    res = b - (double)n * p;
    if (n) {
        R = romberg(a, p, f, eps / (double)n, l, t);
    } R = R *(double)n + romberg(0.0, res, f, eps, l, t);

    return R / 100.0;
}

// 其实不妨先考虑用复化 Simpson 公式
// S = h / 6 * [f(A) + 4 * ∑f(Xk+1/2) + 2 * ∑f(Xk) + f(B)]; k = 0..n-1
```

多项式求根(牛顿法)

```
/* 牛顿法解多项式的根
   输入：多项式系数 c[]，多项式度数 n，求在[a,b]间的根
   输出：根
   要求保证[a,b]间有根
*/
#include <cmath>
#include <cstdlib>

double f(int m, double c[], double x) {
    int i;
    double p = c[m];
    for (i = m; i > 0; i--) {
        p = p * x + c[i - 1];
    }
    return p;
}

int newton(double x0, double * r, double c[], double cp[], int n, double a, double
```

```
b, double eps) {
    const int MAX_ITERATION = 1000;
    int i = 1;
    double x1, x2, fp, eps2 = eps / 10.0;
    x1 = x0;
    while (i < MAX_ITERATION) {
        x2 = f(n, c, x1);
        fp = f(n - 1, cp, x1);
        if ((fabs(fp) < 0.000000001) && (fabs(x2) > 1.0)) {
            return 0;
        }
        x2 = x1 - x2 / fp;
        if (fabs(x1 - x2) < eps2) {
            if (x2 < a || x2 > b) {
                return 0;
            }
            *r = x2;
            return 1;
        }
        x1 = x2;
        i++;
    }
    return 0;
}

double polynomialRoot(double c[], int n, double a, double b, double eps) {
    double * cp;
    int i;
    double root;

    cp = (double *) calloc(n, sizeof(double));
    for (i = n - 1; i >= 0; i--) {
        cp[i] = (i + 1) * c[i + 1];
    }
    if (a > b) {
        root = a;
        a = b;
        b = root;
    }
    if ((!newton(a, &root, c, cp, n, a, b, eps)) && (!newton(b, &root, c, cp, n, a,
 b, eps))) {
        newton((a + b) * 0.5, &root, c, cp, n, a, b, eps);
    }
    free(cp);
    if (fabs(root) < eps) {
        return fabs(root);
    } else {
```

```
        return root;
    }
}
```

周期性方程(追赶法)

```
/*  追赶法解周期性方程
    周期性方程定义: | a1 b1 c1 ...                  |          =  x1
                    |    a2 b2 c2 ...              |          =  x2
                    |       ...                    | *   X    =  ...
                    | cn-1 ...          an-1 bn-1  |          =  xn-1
                    | bn   cn                 an   |          =  xn
    输入: a[],b[],c[],x[]
    输出: 求解结果X在x[]中
*/

void run() {
    c[0] /= b[0];
    a[0] /= b[0];
    x[0] /= b[0];
    for (int i = 1; i < N - 1; i++) {
        double temp = b[i] - a[i] * c[i - 1];
        c[i] /= temp;
        x[i] = (x[i] - a[i] * x[i - 1]) / temp;
        a[i] = -a[i] * a[i - 1] / temp;
    }
    a[N - 2] = -a[N - 2] - c[N - 2];
    for (int i = N - 3; i >= 0; i--) {
        a[i] = -a[i] - c[i] * a[i + 1];
        x[i] -= c[i] * x[i + 1];
    }
    x[N - 1] -= (c[N - 1] * x[0] + a[N - 1] * x[N - 2]);
    x[N - 1] /= (c[N - 1] * a[0] + a[N - 1] * a[N - 2] + b[N - 1]);
    for (int i = N - 2; i >= 0; i--) {
        x[i] += a[i] * x[N - 1];
    }
}
```

图论－NP搜索

最大团

```
//最大团
//返回最大团大小和一个方案,传入图的大小 n 和邻接阵 mat
//mat[i][j]为布尔量
const int MAXN = 60;

void clique(int n, int * u, const bool mat[][MAXN], int size, int & maxs, int & bb,
 int * res, int * rr, int * c) {
    int i, j, vn, v[MAXN];
    if (n) {
        if (size + c[u[0]] <= maxs) {
            return ;
        }
        for (i = 0; i < n + size - maxs && i < n; ++i) {
            for (j = i + 1, vn = 0; j < n; ++j) {
                if (mat[u[i]][u[j]]) {
                    v[vn++] = u[j];
                }
            }
            rr[size] = u[i];
            clique(vn, v, mat, size + 1, maxs, bb, res, rr, c);
            if (bb) {
                return ;
            }
        }
    } else if (size > maxs) {
        maxs = size;
        for (i = 0; i < size; ++i) {
            res[i] = rr[i];
        }
        bb = 1;
    }
}

int maxclique(int n, const bool mat[][MAXN], int * ret) {
    int maxs = 0, bb, c[MAXN], i, j;
    int vn, v[MAXN], rr[MAXN];
    for (c[i = n - 1] = 0; i >= 0; --i) {
        vn = 0;
        for (j = i + 1; j < n; ++j) {
            if (mat[i][j]) {
                v[vn++] = j;
            }
```

```
        }
        bb = 0;
        rr[0] = i;
        clique(vn, v, mat, 1, maxs, bb, ret, rr, c);
        c[i] = maxs;
    }
    return maxs;
}
```

最大团(n<64)(faster)

```
/**
 * WishingBone's ACM/ICPC Routine Library
 *
 * maximum clique solver
 /

// 不知道怎么用……

#include <vector>

using std::vector;

// clique solver calculates both size and consitution of maximum clique
// uses bit operation to accelerate searching
// graph size limit is 63, the graph should be undirected
// can optimize to calculate on each component, and sort on vertex degrees
// can be used to solve maximum independent set
class clique {
public:
    static const long long ONE = 1;
    static const long long MASK = (1 << 21) - 1;
    char bits;
    int n, size, cmax[63];
    long long mask[63], cons;
    // initiate lookup table
    clique() {
        bits = new char[1 << 21];
        bits[0] = 0;
        for (int i = 1; i < 1 << 21; ++i) bits[i] = bits[i >> 1] + (i & 1);
    }
    ~clique() {
        delete bits;
    }
    // search routine
    bool search(int step, int size, long long more, long long con);
    // solve maximum clique and return size
    int sizeClique(vector<vector<int> >& mat);
```

```
    // solve maximum clique and return constitution
    vector<int> consClique(vector<vector<int> >& mat);
};

// search routine
// step is node id, size is current solution, more is available mask, cons is
// constitution mask
bool clique::search(int step, int size, long long more, long long cons) {
    if (step >= n) {
        // a new solution reached
        this->size = size;
        this->cons = cons;
        return true;
    }
    long long now = ONE << step;
    if ((now & more) > 0) {
        long long next = more & mask[step];
        if (size + bits[next & MASK] + bits[(next >> 21) & MASK] + bits[next >>
42] >= this->size
                && size + cmax[step] > this->size) {
            // the current node is in the clique
            if (search(step + 1, size + 1, next, cons | now)) return true;
        }
    }
    long long next = more & ~now;
    if (size + bits[next & MASK] + bits[(next >> 21) & MASK] + bits[next >> 42]
> this->size) {
        // the current node is not in the clique
        if (search(step + 1, size, next, cons)) return true;
    }
    return false;
}

// solve maximum clique and return size
int clique::sizeClique(vector<vector<int> >& mat) {
    n = mat.size();
    // generate mask vectors
    for (int i = 0; i < n; ++i) {
        mask[i] = 0;
        for (int j = 0; j < n; ++j) if (mat[i][j] > 0) mask[i] |= ONE << j;
    }
    size = 0;
    for (int i = n - 1; i >= 0; --i) {
        search(i + 1, 1, mask[i], ONE << i);
        cmax[i] = size;
    }
    return size;
```

```
}

// solve maximum clique and return constitution
// calls sizeClique and restore cons
vector<int> clique::consClique(vector<vector<int> >& mat) {
    sizeClique(mat);
    vector<int> ret;
    for (int i = 0; i < n; ++i) if ((cons & (ONE << i)) > 0) ret.push_back(i);
    return ret;
}
```

最大团

```
const int maxn = 50;

void clique(int n, int mat[][maxn], int num, int U[], int size, int C[], int& _max,
 int ok) {
    int i, j, k, tmp[maxn];
    if (num == 0) {
        if (size > _max) { ok = 1; _max = size; }
        return;
    }
    for (i = 0; i < num && !ok; ++i) {
        if (size + num  - i <= _max) return;
        if (size + C[U[i]] <= _max) return;
        for (k = 0, j = i + 1; j < num; ++j) if (mat[U[i]][U[j]])
            tmp[k++] = U[j];
        clique(n, mat, k, tmp, size + 1, C, _max, ok);
    }
}

int max_clique(int n, int mat[][maxn]) {
    int i, j, k, U[maxn], C[maxn], _max;
    for (_max = 0, i = n - 1; i >= 0; --i) {
        for (k = 0, j = i + 1; j < n; ++j) if (mat[i][j])
            U[k++] = j;
        clique(n, mat, k, U, 1, C, _max, 0);
        C[i] = _max;
    } return _max;
}
```

带权最大团

```
/* wclique.c exact algorithm for finding one maximum-weight
   clique in an arbitrary graph,
   10.2.2000, Patric R. J. Ostergard,
   patric.ostergard@hut.fi /

/ compile: gcc wclique.c -o wclique -O2 /
```

```
/ usage: wclique infile /

/ infile format: see http://www.tcs.hut.fi/~pat/wclique.html */

#include <stdio.h>
#include <sys/times.h>
#include <sys/types.h>

#define INT_SIZE (8*sizeof(int))
#define TRUE 1
#define FALSE 0
#define MAX_VERTEX 2000 /* maximum number of vertices /
#define MAX_WEIGHT 1000000 / maximum weight of vertex /
#define is_edge(a,b) (bit[a][b/INT_SIZE]&(mask[b%INT_SIZE]))

int Vnbr,Enbr;          / number of vertices/edges /
int clique[MAX_VERTEX]; / table for pruning /
int bit[MAX_VERTEX][MAX_VERTEX/INT_SIZE+1];
int wt[MAX_VERTEX];

int pos[MAX_VERTEX];    / reordering function /
int set[MAX_VERTEX];    / current clique /
int rec[MAX_VERTEX];    / best clique so far /
int record;         / weight of best clique /
int rec_level;          / # of vertices in best clique /

unsigned mask[INT_SIZE];
void graph();           / reads graph */

struct tms bf;
int timer1;
double timer11;

main (argc,argv)
int argc;
char *argv[];
{
  int i,j,k,p;
  int min_wt,max_nwt,wth;
  int new[MAX_VERTEX],used[MAX_VERTEX];
  int nwt[MAX_VERTEX];
  int count;
  FILE infile;

  / read input /
  if(argc < 2) {
```

```
    printf("Usage: wclique infile\n");
    exit(1);
  }
  if((infile=fopen(argv[1],"r"))==NULL)
    fileerror();

  / initialize mask /
  mask[0] = 1;
  for(i=1;i<INT_SIZE;i++)
    mask[i] = mask[i-1]<<1;

  / read graph /
  graph(infile);

  / "start clock" /
  times(&bf);
  timer1 = bf.tms_utime;

  / order vertices /
  for(i=0;i<Vnbr;i++) {
    nwt[i] = 0;
    for(j=0;j<Vnbr;j++)
      if (is_edge(i,j)) nwt[i] += wt[j];
  }
  for(i=0;i<Vnbr;i++)
    used[i] = FALSE;
  count = 0;
  do {
     min_wt = MAX_WEIGHT+1; max_nwt = -1;
     for(i=Vnbr-1;i>=0;i--)
       if((!used[i])&&(wt[i]<min_wt))
         min_wt = wt[i];
     for(i=Vnbr-1;i>=0;i--) {
       if(used[i]||(wt[i]>min_wt)) continue;
       if(nwt[i]>max_nwt) {
         max_nwt = nwt[i];
         p = i;
       }
     }
     pos[count++] = p;
     used[p] = TRUE;
     for(j=0;j<Vnbr;j++)
       if((!used[j])&&(j!=p)&&(is_edge(p,j)))
         nwt[j] -= wt[p];
  } while(count<Vnbr);

  /* main routine */
```

```
    record = 0;
    wth = 0;
    for(i=0;i<Vnbr;i++) {
        wth += wt[pos[i]];
        sub(i,pos,0,0,wth);
        clique[pos[i]] = record;
        times(&bf);
        timer11 = (bf.tms_utime - timer1)/100.0;
        printf("level = %3d(%d) best = %2d time = %8.2f\n",i+1,Vnbr,record,timer11);
    }
    printf("Record: ");
    for(i=0;i<rec_level;i++)
      printf ("%d ",rec[i]);
    printf ("\n");
}

int sub(ct,table,level,weight,l_weight)
int ct,level,weight,l_weight;
int *table;
{
    register int i,j,k;
    int best;
    int curr_weight,left_weight;
    int newtable[MAX_VERTEX];
    int *p1,*p2;

    if(ct<=0) { /* 0 or 1 elements left; include these */
      if(ct==0) {
        set[level++] = table[0];
        weight += l_weight;
      }
      if(weight>record) {
        record = weight;
        rec_level = level;
        for (i=0;i<level;i++) rec[i] = set[i];
      }
      return 0;
    }
    for(i=ct;i>=0;i--) {
      if((level==0)&&(i<ct)) return 0;
      k = table[i];
      if((level>0)&&(clique[k]<=(record-weight))) return 0;  /* prune */
      set[level] = k;
      curr_weight = weight+wt[k];
      l_weight -= wt[k];
      if(l_weight<=(record-curr_weight)) return 0; /* prune */
      p1 = newtable;
```

```
    p2 = table;
    left_weight = 0;
    while (p2<table+i) {
      j = *p2++;
      if(is_edge(j,k)) {
    *p1++ = j;
        left_weight += wt[j];
      }
    }
    if(left_weight<=(record-curr_weight)) continue;
    sub(p1-newtable-1,newtable,level+1,curr_weight,left_weight);
  }
  return 0;
}

void graph(fp)
FILE fp;
{
  register int i,j,k;
  int weight,degree,entry;

  if(!fscanf(fp,"%d %d\n",&Vnbr,&Enbr))
    fileerror();
  for(i=0;i<Vnbr;i++)     / empty graph table /
    for(j=0;j<Vnbr/INT_SIZE+1;j++)
      bit[i][j] = 0;
  for(i=0;i<Vnbr;i++) {
    if(!fscanf(fp,"%d %d",&weight,&degree))
      fileerror();
    wt[i] = weight;
    for(j=0;j<degree;j++) {
      if(!fscanf(fp,"%d",&entry))
        fileerror();
      bit[i][entry/INT_SIZE] |= mask[entry%INT_SIZE]; / record edge */
    }
  }
  fclose(fp);
}

int fileerror()
{
  printf("Error in graph file\n");
  exit();
}
```

图论—连通性

无向图关键点(dfs 邻接阵)

```
//无向图的关键点,dfs 邻接阵形式,O(n^2)
//返回关键点个数,key[]返回点集
//传入图的大小 n 和邻接阵 mat,不相邻点边权 0
const int MAXN = 110;

void search(int n, const bool mat[][MAXN], int * dfn, int * low, int now, int & ret
, int * key, int & cnt, int root, int & rd, int * bb) {
    dfn[now] = low[now] = ++cnt;
    for (int i = 0; i < n; i++) {
        if (mat[now][i]) {
            if (!dfn[i]) {
                search(n, mat, dfn, low, i, ret, key, cnt, root, rd, bb);
                if (low[i] < low[now]) {
                    low[now] = low[i];
                }
                if (low[i] >= dfn[now]) {
                    if (now != root && !bb[now]) {
                        key[ret++] = now;
                        bb[now] = 1;
                    } else if (now == root) {
                        rd++;
                    }
                }
            } else if (dfn[i] < low[now]) {
                low[now] = dfn[i];
            }
        }
    }
}

int keyVertex(int n, const bool mat[][MAXN], int * key) {
    int ret = 0, i, cnt, rd, dfn[MAXN], low[MAXN], bb[MAXN];
    for (i = 0; i < n; dfn[i++] = bb[i] = 0);
    for (cnt = i = 0; i < n; i++) {
        if (!dfn[i]) {
            rd = 0;
            search(n, mat, dfn, low, i, ret, key, cnt, i, rd, bb);
            if (rd > 1 && !bb[i]) {
                key[ret++] = i;
                bb[i] = 1;
            }
        }
```

```
    }
    return ret;
}
```

无向图关键边(dfs 邻接阵)

```
//无向图的关键边,dfs 邻接阵形式,O(n^2)
//返回关键边条数,key[][2]返回边集
//传入图的大小 n 和邻接阵 mat,不相邻点边权 0
const int MAXN = 100;

void search(int n, const bool mat[][MAXN], int * dfn, int * low, int now, int & cnt
, int key[][2]) {
    int i;
    for (low[now] = dfn[now], i = 0; i < n; i++) {
        if (mat[now][i]) {
            if (!dfn[i]) {
                dfn[i] = dfn[now] + 1;
                search(n, mat, dfn, low, i, cnt, key);
                if (low[i] > dfn[now]) {
                    key[cnt][0] = i, key[cnt++][1] = now;
                }
                if (low[i] < low[now]) {
                    low[now] = low[i];
                }
            } else if (dfn[i] < dfn[now] - 1 && dfn[i] < low[now]) {
                low[now] = low[i];
            }
        }
    }
}

int keyEdge(int n, const bool mat[][MAXN], int key[][2]) {
    int ret = 0, i, dfn[MAXN], low[MAXN];
    for (i = 0; i < n; dfn[i++] = 0)
        ;
    for (i = 0; i < n; i++) {
        if (!dfn[i]) {
            dfn[i] = 1;
            search(n, mat, dfn, low, i, ret, key);
        }
    }
    return ret;
}
```

无向图的块(bfs 邻接阵)

```
//无向图的块,dfs 邻接阵形式,O(n^2)
//每产生一个块调用 dummy
```

```
//传入图的大小 n 和邻接阵 mat,不相邻点边权 0
const int MAXN = 100;

void dummy(int n, int * a) {
    //...
}

void search(int n, const bool mat[][MAXN], int * dfn, int * low, int now, int & cnt
, int * st, int & sp) {
    int i, m, a[MAXN];
    dfn[st[sp++] = now] = low[now] = ++cnt;
    for (i = 0; i < n; i++) {
        if (mat[now][i]) {
            if (!dfn[i]) {
                search(n, mat, dfn, low, i, cnt, st, sp);
                if (low[i] < low[now]) {
                    low[now] = low[i];
                }
                if (low[i] >= dfn[now]) {
                    for (st[sp] =  - 1, a[0] = now, m = 1; st[sp] != i; a[m++] = st
[--sp])
                        ;
                    dummy(m, a);
                }
            } else if (dfn[i] < low[now]) {
                low[now] = dfn[i];
            }
        }
    }
}

void block(int n, const bool mat[][MAXN]) {
    int i, cnt, dfn[MAXN], low[MAXN], st[MAXN], sp = 0;
    for (i = 0; i < n; dfn[i++] = 0);
    for (cnt = i = 0; i < n; i++) {
        if (!dfn[i]) {
            search(n, mat, dfn, low, i, cnt, st, sp);
        }
    }
}
```

无向图连通分支(dfs/bfs 邻接阵)

```
//无向图连通分支,dfs 邻接阵形式,O(n^2)
//返回分支数,id 返回 1..分支数的值
//传入图的大小 n 和邻接阵 mat,不相邻点边权 0
const int MAXN = 100;
```

```
void floodfill(int n, const bool mat[][MAXN], int * id, int now, int tag) {
    int i;
    for (id[now] = tag, i = 0; i < n; i++) {
        if (!id[i] && mat[now][i]) {
            floodfill(n, mat, id, i, tag);
        }
    }
}

int findComponents(int n, const bool mat[][MAXN], int * id) {
    int ret, i;
    for (i = 0; i < n; id[i++] = 0);
    for (ret = i = 0; i < n; i++) {
        if (!id[i]) {
            floodfill(n, mat, id, i, ++ret);
        }
    }
    return ret;
}

//无向图连通分支,bfs 邻接阵形式,O(n^2)
//返回分支数,id 返回 1..分支数的值
//传入图的大小 n 和邻接阵 mat,不相邻点边权 0
const int MAXN = 100;

int findComponents(int n, const bool mat[][MAXN], int * id) {
    int ret, k, i, j, m;
    for (k = 0; k < n; id[k++] = 0);
    for (ret = k = 0; k < n; k++) {
        if (!id[k]) {
            for (id[k] = -1, ret++, m = 1; m;) {
                for (m = i = 0; i < n; i++) {
                    if (id[i] == -1) {
                        for (m++, id[i] = ret, j = 0; j < n; j++) {
                            if (!id[j] && mat[i][j]) {
                                id[j] = -1;
                            }
                        }
                    }
                }
            }
        }
    }
    return ret;
}
```

有向图强连通分支(dfs 邻接阵)

```
//有向图强连通分支,dfs 邻接阵形式,O(n^2)
//返回分支数,id 返回 1..分支数的值
//传入图的大小 n 和邻接阵 mat,不相邻点边权 0
#include <vector>
using namespace std;
const int MAXN = 1000;
vector < int > loc1[MAXN], loc2[MAXN];

void dfs(int * visit, int now, int & time, int * ft) {
    visit[now] = 1;
    for (int i = 0; i < loc1[now].size(); i++) {
        if (!visit[loc1[now][i]]) {
            dfs(visit, loc1[now][i], time, ft);
        }
    }
    ft[time++] = now;
}

void go(int * visit, int now, int * id, int ret) {
    visit[now] = 1;
    id[now] = ret;
    for (int i = 0; i < loc2[now].size(); i++) {
        if (!visit[loc2[now][i]]) {
            go(visit, loc2[now][i], id, ret);
        }
    }
}

int findComponents(int n, const bool mat[][MAXN], int * id) {
    int visit[MAXN];
    int ft[MAXN];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (mat[i][j]) {
                loc1[i].push_back(j);
                loc2[j].push_back(i);
            }
        }
    }
    for (int i = 0; i < n; i++) {
        visit[i] = 0;
    }
    int time = 0;
    for (int i = 0; i < n; i++) {
        if (!visit[i]) {
            dfs(visit, i, time, ft);
        }
```

```
    }
    for (int i = 0; i < n; i++) {
        visit[i] = 0;
        id[i] = 0;
    }
    int ret = 0;
    for (int i = n - 1; i >= 0; i--) {
        if (!visit[ft[i]]) {
            ret++;
            go(visit, ft[i], id, ret);
        }
    }
    return ret;
}
```

有向图最小点基(邻接阵)

```
//有向图最小点基,邻接阵形式,O(n^2)
//返回电集大小和点集
//传入图的大小 n 和邻接阵 mat,不相邻点边权 0
//需要调用强连通分支
const int MAXN = 100;

int baseVertex(int n, const bool mat[][MAXN], int * sets) {
    int ret = 0, id[MAXN], v[MAXN], i, j;
    j = findComponents(n, mat, id);
    for (i = 0; i < j; v[i++] = 1)
        ;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (id[i] != id[j] && mat[i][j]) {
                v[id[j] - 1] = 0;
            }
        }
    }
    for (i = 0; i < n; i++) {
        if (v[id[i] - 1]) {
            v[id[sets[ret++] = i] - 1] = 0;
        }
    }
    return ret;
}
```

图论－匹配

二分图最大匹配(hungary 邻接表)

```
//二分图最大匹配,hungary 算法,邻接表形式,复杂度 O(m*e)
//返回最大匹配数,传入二分图大小 m,n 和邻接表 list(只需一边)
//match1,match2 返回一个最大匹配,未匹配顶点 match 值为-1
#include <cstring>
const int MAXN = 310;
#define _clr(x) memset(x, 0xff, sizeof(int) * MAXN)

struct Edge {
    int from, to;
    Edge * next;
};

int hungary(int m, int n, Edge * list[], int * match1, int * match2) {
    int s[MAXN + 1], t[MAXN], p, q, ret = 0, i, j, k;
    Edge * e;
    _clr(match1);
    _clr(match2);
    for (i = 0; i < m; ret += (match1[i++] >= 0)) {
        _clr(t);
        for (s[p = q = 0] = i; p <= q && match1[i] < 0; p++) {
            for (e = list[k = s[p]]; e && match1[i] < 0; e = e->next) {
                if (t[j = e->to] < 0) {
                    s[++q] = match2[j];
                    t[j] = k;
                    if (s[q] < 0) {
                        for (p = j; p >= 0; j = p) {
                            match2[j] = k = t[j];
                            p = match1[k];
                            match1[k] = j;
                        }
                    }
                }
            }
        }
    }
    return ret;
}
```

二分图最大匹配(hungary 邻接阵)

```
//二分图最大匹配,hungary 算法,邻接阵形式,复杂度 O(m*m*n)
//返回最大匹配数,传入二分图大小 m,n 和邻接阵 mat,非零元素表示有边
//match1,match2 返回一个最大匹配,未匹配顶点 match 值为-1
```

```
#include <cstring>
const int MAXN = 310;
#define _clr(x) memset(x, 0xff, sizeof(int) * MAXN)

int hungary(int m, int n, const bool mat[][MAXN], int * match1, int * match2) {
    int s[MAXN + 1], t[MAXN], p, q, ret = 0, i, j, k;
    _clr(match1);
    _clr(match2);
    for (i = 0; i < m; ret += (match1[i++] >= 0)) {
        _clr(t);
        for (s[p = q = 0] = i; p <= q && match1[i] < 0; p++) {
            k = s[p];
            for (j = 0; j < n && match1[i] < 0; j++) {
                if (mat[k][j] && t[j] < 0) {
                    s[++q] = match2[j];
                    t[j] = k;
                    if (s[q] < 0) {
                        for (p = j; p >= 0; j = p) {
                            match2[j] = k = t[j];
                            p = match1[k];
                            match1[k] = j;
                        }
                    }
                }
            }
        }
    }
    return ret;
}
```

二分图最大匹配(hungary 邻接表形式,邻接阵接口)

```
//二分图最大匹配,hungary 算法,邻接表形式,邻接阵接口,复杂度 O(m*e)s
//返回最大匹配数,传入二分图大小 m,n 和邻接阵
//match1,match2 返回一个最大匹配,未匹配顶点 match 值为-1
#include <cstring>
#include <vector>
using namespace std;
const int MAXN = 310;
#define _clr(x) memset(x, 0xff, sizeof(int) * MAXN)

int hungary(int m, int n, const bool mat[][MAXN], int * match1, int * match2) {
    int s[MAXN + 1], t[MAXN], p, q, ret = 0, i, j, k, r;
    vector <int> e[MAXN];
    //生成邻接表(只需一边)
    for (i = 0; i < m; ++i) {
        for (j = 0; j < n; ++j) {
            if (mat[i][j]) {
```

```
                e[i].push_back(j);
            }
        }
    }
    _clr(match1);
    _clr(match2);
    for ( i = 0; i < m; ret += (match1[i++] >= 0)) {
        _clr(t);
        for (s[p = q = 0] = i; p <= q && match1[i] < 0; p++) {
            for (r = 0, k = s[p]; r < e[k].size() && match1[i] < 0; ++r) {
                if (t[j = e[k][r]] < 0) {
                    s[++q] = match2[j];
                    t[j] = k;
                    if (s[q] < 0) {
                        for (p = j; p >= 0; j = p) {
                            match2[j] = k = t[j];
                            p = match1[k];
                            match1[k] = j;
                        }
                    }
                }
            }
        }
    }
    return ret;
}
```

二分图最佳匹配(kuhn_munkras 邻接阵)

```
//二分图最佳匹配,kuhn munkras 算法,邻接阵形式,复杂度 O(m*m*n)
//返回最佳匹配值,传入二分图大小 m,n 和邻接阵 mat,表示权值
//match1,match2 返回一个最佳匹配,未匹配顶点 match 值为-1
//一定注意 m<=n,否则循环无法终止
//最小权匹配可将权值取相反数
#include <cstring>
const int MAXN = 310;
const int INF = 1000000000;
#define _clr(x) memset(x, 0xff, sizeof(int) * n)

int kuhnMunkras(int m, int n, int mat[][MAXN], int * match1, int * match2) {
    int s[MAXN + 1], t[MAXN], l1[MAXN], l2[MAXN], p, q, ret = 0, i, j, k;
    for (i = 0; i < m; i++) {
        l1[i] =  - INF;
        for (j = 0; j < n; j++) {
            l1[i] = mat[i][j] > l1[i] ? mat[i][j]: l1[i];
        }
    }
    for (i = 0; i < n; l2[i++] = 0);
```

```
    _clr(match1);
    _clr(match2);
    for (i = 0; i < m; i++) {
        _clr(t);
        for (s[p = q = 0] = i; p <= q && match1[i] < 0; p++) {
            k = s[p];
            for (j = 0; j < n && match1[i] < 0; j++) {
                if (l1[k] + l2[j] == mat[k][j] && t[j] < 0) {
                    s[++q] = match2[j];
                    t[j] = k;
                    if (s[q] < 0) {
                        for (p = j; p >= 0; j = p) {
                            match2[j] = k = t[j];
                            p = match1[k];
                            match1[k] = j;
                        }
                    }
                }
            }
        }
        if (match1[i] < 0) {
            i--;
            p = INF;
            for (k = 0; k <= q; k++) {
                for (j = 0; j < n; j++) {
                    if (t[j] < 0 && l1[s[k]] + l2[j] - mat[s[k]][j] < p) {
                        p = l1[s[k]] + l2[j] - mat[s[k]][j];
                    }
                }
            }
            for (j = 0; j < n; j++) {
                l2[j] += t[j] < 0 ? 0 : p;
            }
            for (k = 0; k <= q; k++) {
                l1[s[k]] -= p;
            }
        }
    }
    for (i = 0; i < m; i++) {
        ret += mat[i][match1[i]];
    }
    return ret;
}
```

一般图匹配(邻接表)

```
//一般图最大匹配,邻接表形式,复杂度O(n*e)
//返回匹配顶点对数,match返回匹配,未匹配顶点match值为-1
```

```
//传入图的顶点数 n 和邻接表 list
#include <cstring>
const int MAXN = 100;

struct Edge {
    int from, to;
    Edge * next;
};

int aug(int n, Edge * list[], int * match, int * v, int now) {
    int t, ret = 0;
    Edge * e;
    v[now] = 1;
    for (e = list[now]; e; e = e->next) {
        if (!v[t = e->to]) {
            if (match[t] < 0) {
                match[now] = t;
                match[t] = now;
                ret = 1;
            } else {
                v[t] = 1;
                if (aug(n, list, match, v, match[t])) {
                    match[now] = t;
                    match[t] = now;
                    ret = 1;
                }
            }
            if (ret) {
                break;
            }
        }
    }
    v[now] = 0;
    return ret;
}

int graphMatch(int n, Edge * list[], int * match) {
    int v[MAXN], i, j;
    for (i = 0; i < n; i++) {
        v[i] = 0, match[i] = -1;
    }
    for (i = 0, j = n; i < n && j >= 2;) {
        if (match[i] < 0 && aug(n, list, match, v, i)) {
            memset(v, 0, sizeof(v));
            i = 0;
            j -= 2;
        } else {
```

```
            i++;
        }
    }
    for (i = j = 0; i < n; i++) {
        j += (match[i] >= 0);
    }
    return j / 2;
}
```

一般图匹配(邻接阵)

```
//一般图最大匹配,邻接阵形式,复杂度O(n^3)
//返回匹配顶点对数,match返回匹配,未匹配顶点match值为-1
//传入图的顶点数n和邻接阵mat
#include <cstring>
const int MAXN = 100;

int aug(int n, const bool mat[][MAXN], int * match, int * v, int now) {
    int i, ret = 0;
    v[now] = 1;
    for (i = 0; i < n; i++) {
        if (!v[i] && mat[now][i]) {
            if (match[i] < 0) {
                match[now] = i;
                match[i] = now;
                ret = 1;
            } else {
                v[i] = 1;
                if (aug(n, mat, match, v, match[i])) {
                    match[now] = i;
                    match[i] = now;
                    ret = 1;
                }
            }
            if (ret) {
                break;
            }
        }
    }
    v[now] = 0;
    return ret;
}

int graphMatch(int n, const bool mat[][MAXN], int * match) {
    int v[MAXN], i, j;
    for (i = 0; i < n; i++) {
        v[i] = 0;
        match[i] = -1;
```

```
    }
    for (i = 0, j = n; i < n && j >= 2;) {
        if (match[i] < 0 && aug(n, mat, match, v, i)) {
            memset(v, 0, sizeof(v)), i = 0, j -= 2;
        } else {
            i++;
        }
    }
    for (i = j = 0; i < n; i++) {
        j += (match[i] >= 0);
    }
    return j / 2;
}
```

一般图匹配(邻接表形式,邻接阵接口)

```
//一般图最大匹配,邻接表形式,复杂度O(n*e)
//返回匹配顶点对数,match返回匹配,未匹配顶点match值为-1
//传入图的顶点数n和邻接表list
#include <vector>
using namespace std;
const int MAXN = 100;

int aug(int n, const vector <int> list[], int * match, int * v, int now) {
    int t, ret = 0, r;
    v[now] = 1;
    //    for (e=list[now];e;e=e->next)
    for (r = 0; r < list[now].size(); ++r) {
        if (!v[t = list[now][r]]) {
            if (match[t] < 0) {
                match[now] = t;
                match[t] = now;
                ret = 1;
            } else {
                v[t] = 1;
                if (aug(n, list, match, v, match[t])) {
                    match[now] = t;
                    match[t] = now;
                    ret = 1;
                }
            }
            if (ret) {
                break;
            }
        }
    }
    v[now] = 0;
    return ret;
```

```
}

int graphMatch(int n, const bool mat[][MAXN], int * match) {
    int v[MAXN], i, j;
    vector <int> list[MAXN];
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (mat[i][j]) {
                list[i].push_back(j);
            }
        }
    }
    for (i = 0; i < n; i++) {
        v[i] = 0, match[i] = -1;
    }
    for (i = 0, j = n; i < n && j >= 2;) {
        if (match[i] < 0 && aug(n, list, match, v, i)) {
            memset(v, 0, sizeof(v)), i = 0, j -= 2;
        } else {
            i++;
        }
    }
    for (i = j = 0; i < n; i++) {
        j += (match[i] >= 0);
    }
    return j / 2;
}
```

图论－网络流

最大流(Dinic)

```
/*
    最大流 Dinic 算法 by dd_engi
    1.算法被封装成了一个 struct。
    2.struct 需要在全局变量中声明或者 new 出来，千万不要声明成栈上的局部变量。
    3.每次使用前，dinic.init(S,T)给定源与汇的编号，然后用 dinic.add_edge(x,y,w)添加每条有容量的边。
    4.调用 dinic.flow()进行计算，返回最大流的值；每条边的流量有储存在 edge.f 里。
    5.同一个 struct 可以处理多组数据，但每次都要先 init。
    6.不需要知道总点数，点的编号可以不连续，但是所有的编号都需要在[0,MAXN]之间。
    7.可处理多重边。
    8.dinic.cut()是一个附送的功能，调用 flow()后，可用它求出最小割中的 T 集。返回 T 集的大小，元素保存在传入的数组中。
*/
#include <cstdio>
#include <cstring>
#include <climits>
using namespace std;

const int MAXN=22000,MAXM=440000;
struct Dinic{
    struct edge{
        int x,y;//两个顶点
        int c;//容量
        int f;//当前流量
        edge *next,*back;//下一条边，反向边
        edge(int x,int y,int c,edge* next):x(x),y(y),c(c),f(0),next(next),back(0){}
        void* operator new(size_t, void *p){return p;}
    }*E[MAXN],*data;//E[i]保存顶点 i 的边表
    char storage[2*MAXM*sizeof(edge)];
    int S,T;//源、汇

    int Q[MAXN];//DFS 用到的 queue
    int D[MAXN];//距离标号，-1 表示不可达
    void DFS(){
        memset(D,-1,sizeof(D));
        int head=0,tail=0;
        Q[tail++]=S;
        D[S]=0;
        for(;;){
            int i=Q[head++];
            for(edge* e=E[i];e;e=e->next){
                if(e->c==0)continue;
```

```
                int j=e->y;
                if(D[j]==-1){
                    D[j]=D[i]+1;
                    Q[tail++]=j;
                    if(j==T)return;
                }
            }
            if(head==tail)break;
        }
    }
    edge* cur[MAXN];//当前弧
    edge* path[MAXN];//当前找到的增广路
    int flow(){
        int res=0;//结果，即总流量
        int path_n;//path 的大小
        for(;;){
            DFS();
            if(D[T]==-1)break;
            memcpy(cur,E,sizeof(E));
            path_n=0;
            int i=S;
            for(;;){
                if(i==T){//已找到一条增广路，增广之
                    int mink=0;
                    int delta=INT_MAX;
                    for(int k=0;k<path_n;++k){
                        if(path[k]->c < delta){
                            delta = path[k]->c;
                            mink=k;
                        }
                    }
                    for(int k=0;k<path_n;++k){
                        path[k]->c -= delta;
                        path[k]->back->c += delta;
                    }
                    path_n=mink;//回退
                    i=path[path_n]->x;
                    res+=delta;
                }
                edge* e;
                for(e=cur[i];e;e=e->next){
                    if(e->c==0)continue;
                    int j=e->y;
                    if(D[i]+1==D[j])break;//找到一条弧，加到路径里
                }
                cur[i]=e;//当前弧结构，访问过的不能增广的弧不会再访问
                if(e){
```

```
                    path[path_n++]=e;
                    i=e->y;
                }
                else{//该节点已没有任何可增广的弧，从图中删去，回退一步
                    D[i]=-1;
                    if(path_n==0)break;
                    path_n--;
                    i=path[path_n]->x;
                }
            }
        }
        return res;
    }
    int cut(int* s){
        int rst=0;
        for(int i=0;i<MAXN;++i)
            if(D[i]==-1&&E[i])
                s[rst++]=i;
        return rst;
    }
    void init(int _S,int _T){
        S=_S,T=_T;
        data=(edge*)storage;
        memset(E,0,sizeof(E));
    }
    void add_edge(int x,int y,int w){//加进一条 x 至 y 容量为 w 的边，需要保证 0<=x,y<MAXN,
0<w<=INT_MAX
        E[x]=new((void*)data++) edge(x,y,w,E[x]);
        E[y]=new((void*)data++) edge(y,x,0,E[y]);
        E[x]->back = E[y];
        E[y]->back = E[x];
    }
};


/**** 示范用法 ****/
Dinic dinic;
int main(){
    int N,M;
    while(2==scanf("%d%d",&N,&M)){
        int rst=0;
        int S=0,T=N+1;
        dinic.init(S,T);
        for(int i=1;i<=N;++i){
            int a,b;
            scanf("%d%d",&a,&b);
            dinic.add_edge(S,i,a);
```

```
            dinic.add_edge(i,T,b);
        }
        for(int i=0;i<M;++i){
            int x,y,w;
            scanf("%d%d%d",&x,&y,&w);
            dinic.add_edge(x,y,w);
            dinic.add_edge(y,x,w);
        }
        rst=dinic.flow();
        printf("%d\n",rst);
    }
}
```

最大流(Dinic)

```
// 大数据推荐

#include <cstdio>
#include <cstring>
#include <climits>
#include <iostream>
using namespace std;

//要清空的有E,data 每次要赋值为 start
//MAXM 应为实际边数的 2 倍 因为有反向边
const int MAXN=500,MAX=1000000000;
struct edge{
    int x,y;//两个顶点
    int c;//容量
    int f;//当前流量
    edge *next,*back;//下一条边，反向边
    edge(){}
    edge(int x,int y,int c,edge* next):x(x),y(y),c©,f(0),next(next),back(0){}
    void* operator new(size_t, void *p){
        return p;
    }
}*E[MAXN];//E[i]保存顶点 i 的边表
edge* start, *data; //data 为实际申请的空间,start 记录 data 开始的指针,每 case 将 data 赋值为 start,可以不用清空
/*
    Sample Usage:
    data=new edge[MAXM];
    start = data;
/
int Q[MAXN];//DFS 用到的 queue
int D[MAXN];//距离标号，-1 表示不可达
void DFS(int S, int T)
{
```

```
    memset(D,-1,sizeof(D));
    int head=0,tail=0;
    Q[tail++]=S;
    D[S]=0;
    for(;;){
        int i=Q[head++];
        for(edge e=E[i];e;e=e->next){
            if(e->c==0)continue;
            int j=e->y;
            if(D[j]==-1){
                D[j]=D[i]+1;
                Q[tail++]=j;
                if(j==T)return;
            }
        }
        if(head==tail)break;
    }
}
edge* cur[MAXN];//当前弧
edge* path[MAXN];//当前找到的增广路
int flow(int S, int T)
{
    int res=0;//结果，即总流量
    int path_n;//path 的大小
    for(;;){
        DFS(S, T);
        if(D[T]==-1)break;
        memcpy(cur,E,sizeof(E));
        path_n=0;
        int i=S;
        for(;;){
            if(i==T){//已找到一条增广路，增广之
                int mink=0;
                int delta=INT_MAX;
                for(int k=0;k<path_n;++k){
                    if(path[k]->c < delta){
                        delta = path[k]->c;
                        mink=k;
                    }
                }
                for(int k=0;k<path_n;++k){
                    path[k]->c -= delta;
                    path[k]->back->c += delta;
                }
                path_n=mink;//回退
                i=path[path_n]->x;
                res+=delta;
```

```
            }
            edge* e;
            for(e=cur[i];e;e=e->next){
                if(e->c==0)continue;
                int j=e->y;
                if(D[i]+1==D[j])break;//找到一条弧，加到路径里
            }
            cur[i]=e;//当前弧结构，访问过的不能增广的弧不会再访问
            if(e){
                path[path_n++]=e;
                i=e->y;
            }
            else{//该节点已没有任何可增广的弧，从图中删去，回退一步
                D[i]=-1;
                if(path_n==0)break;
                path_n--;
                i=path[path_n]->x;
            }
        }
    }
    return res;
}
inline void AddEdge(int x, int y, int c){
    E[x]=new((void*)data++) edge(x,y,c,E[x]);
    E[y]=new((void*)data++) edge(y,x,0,E[y]);
    E[x]->back = E[y];
    E[y]->back = E[x];
}
int main(){

}
```

最大流(邻接阵)

```
//求网络最大流,邻接阵形式
//返回最大流量,flow 返回每条边的流量
//传入网络节点数 n,容量 mat,源点 source,汇点 sink

const int MAXN = 100;
const int INF = 1000000000;

int maxFlow(int n, const int mat[][MAXN], int source, int sink, int flow[][MAXN]) {
    int pre[MAXN], que[MAXN], d[MAXN], p, q, t, i, j;
    if (source == sink) {
        return INF;
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; flow[i][j++] = 0);
```

```
    }
    while (true) {
        for (i = 0; i < n; pre[i++] = 0);
        pre[t = source] = source + 1;
        d[t] = INF;
        for (p = q = 0; p <= q && !pre[sink]; t = que[p++]) {
            for (i = 0; i < n; i++) {
                if (!pre[i] && (j = mat[t][i] - flow[t][i])) {
                    pre[que[q++] = i] = t + 1;
                    d[i] = d[t] < j ? d[t] : j;
                } else if (!pre[i] && (j = flow[i][t])) {
                    pre[que[q++] = i] = -t - 1;
                    d[i] = d[t] < j ? d[t] : j;
                }
            }
        }
        if (!pre[sink]) {
            break;
        }
        for (i = sink; i != source;) {
            if (pre[i] > 0) {
                flow[pre[i] - 1][i] += d[sink];
                i = pre[i] - 1;
            } else {
                flow[i][-pre[i] - 1] -= d[sink];
                i = -pre[i] - 1;
            }
        }
    }
    for (j = i = 0; i < n; j += flow[source][i++]);
    return j;
}
```

最大流(邻接表)

```
//求网络最大流,邻接表形式
//返回最大流量,flow 返回每条边的流量
//传入网络节点数 n,容量 mat,邻接表 list,源点 source,汇点 sink
//list[i](vector<int>)存放所有以 i 相邻的点,包括反向边!!!
#include <vector>
using namespace std;
const int MAXN = 100;
const int INF = 1000000000;

int maxFlow(int n, const int mat[][MAXN], const vector <int> list[], int source, in
t sink, int flow[][MAXN]) {
    int pre[MAXN], que[MAXN], d[MAXN], p, q, t, i, j, r;
    if (source == sink) {
```

```
        return INF;
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; flow[i][j++] = 0);
    }
    while (true) {
        for (i = 0; i < n; pre[i++] = 0);
        pre[t = source] = source + 1;
        d[t] = INF;
        for (p = q = 0; p <= q && !pre[sink]; t = que[p++]) {
            for (r = 0; r < list[t].size(); ++r) {
                i = list[t][r];
                if (!pre[i] && (j = mat[t][i] - flow[t][i])) {
                    pre[que[q++] = i] = t + 1;
                    d[i] = d[t] < j ? d[t] : j;
                } else if (!pre[i] && (j = flow[i][t])) {
                    pre[que[q++] = i] = -t - 1;
                    d[i] = d[t] < j ? d[t] : j;
                }
            }
        }
        if (!pre[sink]) {
            break;
        }
        for (i = sink; i != source;) {
            if (pre[i] > 0) {
                flow[pre[i] - 1][i] += d[sink];
                i = pre[i] - 1;
            } else {
                flow[i][-pre[i] - 1] -= d[sink];
                i =  - pre[i] - 1;
            }
        }
    }
    for (j = i = 0; i < n; j += flow[source][i++]);
    return j;
}
```

最大流(邻接表形式,邻接阵接口)

```
//求网络最大流,邻接表形式
//返回最大流量,flow 返回每条边的流量
//传入网络节点数 n,容量 mat,源点 source,汇点 sink
#include <vector>
using namespace std;
const int MAXN = 100;
const int INF = 1000000000;
```

```
int maxFlow(int n, const int mat[][MAXN], int source, int sink, int flow[][MAXN]) {
    int pre[MAXN], que[MAXN], d[MAXN], p, q, t, i, j, r;
    vector <int> e[MAXN];
    if (source == sink) {
        return INF;
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; flow[i][j++] = 0);
    }
    //e[i]存放所有以 i 相邻的点,包括反向边!!!
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (mat[i][j] || mat[j][i]) {
                e[i].push_back(j);
            }
        }
    }
    while (true) {
        for (i = 0; i < n; pre[i++] = 0);
        pre[t = source] = source + 1;
        d[t] = INF;
        for (p = q = 0; p <= q && !pre[sink]; t = que[p++]) {
            for (r = 0; r < e[t].size(); ++r) {
                i = e[t][r];
                if (!pre[i] && (j = mat[t][i] - flow[t][i])) {
                    pre[que[q++] = i] = t + 1;
                    d[i] = d[t] < j ? d[t] : j;
                } else if (!pre[i] && (j = flow[i][t])) {
                    pre[que[q++] = i] = -t - 1;
                    d[i] = d[t] < j ? d[t] : j;
                }
            }
        }
        if (!pre[sink]) {
            break;
        }
        for (i = sink; i != source;) {
            if (pre[i] > 0) {
                flow[pre[i] - 1][i] += d[sink];
                i = pre[i] - 1;
            } else {
                flow[i][-pre[i] - 1] -= d[sink];
                i = -pre[i] - 1;
            }
        }
    }
    for (j = i = 0; i < n; j += flow[source][i++]);
```

```
    return j;
}
```

上下界最大流(邻接阵)

```
//求上下界网络最大流,邻接阵形式
//返回最大流量,-1 表示无可行流,flow 返回每条边的流量
//传入网络节点数 n,容量 mat,流量下界 bf,源点 source,汇点 sink
//MAXN 应比最大结点数多 2,无可行流返回-1 时 mat 未复原!

const int MAXN = 100;
const int INF = 1000000000;

int limitMaxFlow(int n, int mat[][MAXN], int bf[][MAXN], int source, int sink, int
flow[][MAXN]) {
    int i, j, sk, ks;
    if (source == sink) {
        return INF;
    }
    mat[n][n + 1] = mat[n + 1][n] = mat[n][n] = mat[n + 1][n + 1] = 0;
    for (i = 0; i < n; i++) {
        mat[n][i] = mat[i][n] = mat[n + 1][i] = mat[i][n + 1] = 0;
        for (j = 0; j < n; j++) {
            mat[i][j] -= bf[i][j];
            mat[n][i] += bf[j][i];
            mat[i][n + 1] += bf[i][j];
        }
    }
    sk = mat[source][sink];
    ks = mat[sink][source];
    mat[source][sink] = mat[sink][source] = INF;
    for (i = 0; i < n + 2; i++) {
        for (j = 0; j < n + 2; flow[i][j++] = 0);
    }
    maxFlow(n + 2, mat, n, n + 1, flow);
    for (i = 0; i < n; i++) {
        if (flow[n][i] < mat[n][i]) {
            return -1;
        }
    }
    flow[source][sink] = flow[sink][source] = 0;
    mat[source][sink] = sk;
    mat[sink][source] = ks;
    maxFlow(n, mat, source, sink, flow);
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            mat[i][j] += bf[i][j];
            flow[i][j] += bf[i][j];
```

```
        }
    }
    for (j = i = 0; i < n; j += flow[source][i++]);
    return j;
}
```

上下界最大流(邻接表)

```
//求上下界网络最大流,邻接表形式
//返回最大流量,-1 表示无可行流,flow 返回每条边的流量
//传入网络节点数 n,容量 mat,流量下界 bf,源点 source,汇点 sink
//MAXN 应比最大结点数多 2,无可行流返回-1 时 mat 未复原!
#include <vector>
using namespace std;

const int MAXN = 100;
const int INF = 1000000000;

int maxFlow(int n, const int mat[][MAXN], int source, int sink, int flow[][MAXN]) {
    int pre[MAXN], que[MAXN], d[MAXN], p, q, t, i, j, r;
    vector < int > e[MAXN];
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (mat[i][j] || mat[j][i]) {
                e[i].push_back(j);
            }
        }
    }
    while (true) {
        for (i = 0; i < n; pre[i++] = 0);
        pre[t = source] = source + 1;
        d[t] = INF;
        for (p = q = 0; p <= q && !pre[sink]; t = que[p++]) {
            for (r = 0; r < e[t].size(); ++r) {
                i = e[t][r];
                if (!pre[i] && (j = mat[t][i] - flow[t][i])) {
                    pre[que[q++] = i] = t + 1;
                    d[i] = d[t] < j ? d[t] : j;
                } else if (!pre[i] && (j = flow[i][t])) {
                    pre[que[q++] = i] = -t - 1;
                    d[i] = d[t] < j ? d[t] : j;
                }
            }
        }
        if (!pre[sink]) {
            break;
        }
        for (i = sink; i != source;) {
```

```
            if (pre[i] > 0) {
                flow[pre[i] - 1][i] += d[sink];
                i = pre[i] - 1;
            } else {
                flow[i][ - pre[i] - 1] -= d[sink];
                i =  - pre[i] - 1;
            }
        }
    }
    for (j = i = 0; i < n; j += flow[source][i++]);
    return j;
}

int limitMaxFlow(int n, int mat[][MAXN], int bf[][MAXN], int source, int sink, int
flow[][MAXN]) {
    int i, j, sk, ks;
    if (source == sink) {
        return INF;
    }
    mat[n][n + 1] = mat[n + 1][n] = mat[n][n] = mat[n + 1][n + 1] = 0;
    for (i = 0; i < n; i++) {
        mat[n][i] = mat[i][n] = mat[n + 1][i] = mat[i][n + 1] = 0;
        for (j = 0; j < n; j++) {
            mat[i][j] -= bf[i][j];
            mat[n][i] += bf[j][i];
            mat[i][n + 1] += bf[i][j];
        }
    }
    sk = mat[source][sink];
    ks = mat[sink][source];
    mat[source][sink] = mat[sink][source] = INF;
    for (i = 0; i < n + 2; i++) {
        for (j = 0; j < n + 2; flow[i][j++] = 0);
    }
    maxFlow(n + 2, mat, n, n + 1, flow);
    for (i = 0; i < n; i++) {
        if (flow[n][i] < mat[n][i]) {
            return  - 1;
        }
    }
    flow[source][sink] = flow[sink][source] = 0;
    mat[source][sink] = sk;
    mat[sink][source] = ks;
    maxFlow(n, mat, source, sink, flow);
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            mat[i][j] += bf[i][j];
```

```
            flow[i][j] += bf[i][j];
        }
    }
    for (j = i = 0; i < n; j += flow[source][i++]);
    return j;
}
```

上下界最小流(邻接阵)

```
//求上下界网络最小流,邻接阵形式
//返回 true 表示有可行流,flow 为每条边的流量,minFlow 为最小流量,返回 false 表示无可行流
//传入网络节点数 n,容量 mat,流量下界 bf,源点 source,汇点 sink
//allowNegativeFlow,true 时允许结果流量为负,false 时不允许结果流量为负
//MAXN 应比最大节点数多 4,无可行流返回 false 时 mat 未复原!

const int MAXN = 128;
const int INF = 1000000000;

void maxFlow(int n, const int mat[][MAXN], int source, int sink, int flow[][MAXN],
int remain) {
    int pre[MAXN], que[MAXN], d[MAXN], p, q, t, i, j;
    while (remain) {
        for (i = 0; i < n; pre[i++] = 0);
        pre[t = source] = source + 1;
        d[t] = INF;
        for (p = q = 0; p <= q && !pre[sink]; t = que[p++]) {
            for (i = 0; i < n; i++) {
                if (!pre[i] && (j = mat[t][i] - flow[t][i])) {
                    pre[que[q++] = i] = t + 1;
                    d[i] = d[t] < j ? d[t] : j;
                } else if (!pre[i] && (j = flow[i][t])) {
                    pre[que[q++] = i] = -t - 1;
                    d[i] = d[t] < j ? d[t] : j;
                }
            }
        }
        if (!pre[sink]) {
            break;
        }
        if (remain >= 0) {
            if (d[sink] > remain) {
                d[sink] = remain;
            }
            remain -= d[sink];
        }
        for (i = sink; i != source;) {
            if (pre[i] > 0) {
                flow[pre[i] - 1][i] += d[sink], i = pre[i] - 1;
```

```
            } else {
                flow[i][ - pre[i] - 1] -= d[sink], i =  - pre[i] - 1;
            }
        }
    }
}

bool limitMinFlow(int n, int mat[][MAXN], int bf[][MAXN], int source, int sink, int
 flow[][MAXN], bool allowNegativeFlow, int & minFlow) {
    int i, j;
    const int ss = n;
    const int kk = n + 1;
    for (n += 2, i = 0; i < n; ++i) {
        mat[ss][i] = mat[i][ss] = mat[i][kk] = mat[kk][i] = bf[ss][i] = bf[i][ss] =
 bf[i][kk] = bf[kk][i] = 0;
    }
    mat[ss][source] = mat[sink][kk] = INF;
    mat[n][n + 1] = mat[n + 1][n] = mat[n][n] = mat[n + 1][n + 1] = 0;
    for (i = 0; i < n; i++) {
        mat[n][i] = mat[i][n] = mat[n + 1][i] = mat[i][n + 1] = 0;
        for (j = 0; j < n; j++) {
            mat[i][j] -= bf[i][j];
            mat[n][i] += bf[j][i];
            mat[i][n + 1] += bf[i][j];
        }
    }
    mat[kk][ss] = INF;
    if (allowNegativeFlow) {
        mat[ss][kk] = INF;
    }
    for (i = 0; i < n + 2; i++) {
        for (j = 0; j < n + 2; flow[i][j++] = 0);
    }
    maxFlow(n + 2, mat, n, n + 1, flow,  - 1);
    for (i = 0; i < n; i++) {
        if (flow[n][i] < mat[n][i]) {
            return false;
        }
    }
    flow[ss][kk] = flow[kk][ss] = mat[ss][kk] = mat[kk][ss] = 0;
    if (!allowNegativeFlow) {
        maxFlow(n, mat, ss, kk, flow,  - 1);
        for (j = i = 0; i < n; ++i) {
            j += flow[ss][i] - flow[i][ss];
        }
        if (j < 0) {
            return false;
```

```
        }
        maxFlow(n, mat, kk, ss, flow, j);
    } else {
        maxFlow(n, mat, kk, ss, flow,  - 1);
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            mat[i][j] += bf[i][j];
            flow[i][j] += bf[i][j];
        }
    }
    for (minFlow = i = 0; i < n; ++i) {
        minFlow += flow[ss][i] - flow[i][ss];
    }
    return true;
}
```

上下界最小流(邻接表)

```
//求上下界网络最小流,邻接表形式
//返回最大流量,-1 表示无可行流,flow 返回每条边的流量
//传入网络节点数 n,容量 mat,流量下界 bf,源点 source,汇点 sink
//MAXN 应比最大结点数多 2,无可行流返回-1 时 mat 未复原!
#include <vector>
using namespace std;
const int MAXN = 100;
const int INF = 1000000000;

int maxFlow(int n, const int mat[][MAXN], int source, int sink, int flow[][MAXN]) {
    int pre[MAXN], que[MAXN], d[MAXN], p, q, t, i, j, r;
    vector < int > e[MAXN];
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (mat[i][j] || mat[j][i]) {
                e[i].push_back(j);
            }
        }
    }
    while (true) {
        for (i = 0; i < n; pre[i++] = 0);
        pre[t = source] = source + 1;
        d[t] = INF;
        for (p = q = 0; p <= q && !pre[sink]; t = que[p++]) {
            for (r = 0; r < e[t].size(); ++r) {
                i = e[t][r];
                if (!pre[i] && (j = mat[t][i] - flow[t][i])) {
                    pre[que[q++] = i] = t + 1;
                    d[i] = d[t] < j ? d[t] : j;
                } else if (!pre[i] && (j = flow[i][t])) {
```

```
                    pre[que[q++] = i] = -t - 1;
                    d[i] = d[t] < j ? d[t] : j;
                }
            }
        }
        if (!pre[sink]) {
            break;
        }
        for (i = sink; i != source;) {
            if (pre[i] > 0) {
                flow[pre[i] - 1][i] += d[sink];
                i = pre[i] - 1;
            } else {
                flow[i][-pre[i] - 1] -= d[sink];
                i = -pre[i] - 1;
            }
        }
    }
    for (j = i = 0; i < n; j += flow[source][i++]);
    return j;
}

int limitMinFlow(int n, int mat[][MAXN], int bf[][MAXN], int source, int sink, int
flow[][MAXN]) {
    int i, j, sk, ks;
    if (source == sink) {
        return INF;
    }
    mat[n][n + 1] = mat[n + 1][n] = mat[n][n] = mat[n + 1][n + 1] = 0;
    for (i = 0; i < n; i++) {
        mat[n][i] = mat[i][n] = mat[n + 1][i] = mat[i][n + 1] = 0;
        for (j = 0; j < n; j++) {
            mat[i][j] -= bf[i][j];
            mat[n][i] += bf[j][i];
            mat[i][n + 1] += bf[i][j];
        }
    }
    sk = mat[source][sink];
    ks = mat[sink][source];
    mat[source][sink] = mat[sink][source] = INF;
    for (i = 0; i < n + 2; i++) {
        for (j = 0; j < n + 2; flow[i][j++] = 0);
    }
    maxFlow(n + 2, mat, n, n + 1, flow);
    for (i = 0; i < n; i++) {
        if (flow[n][i] < mat[n][i]) {
            return  - 1;
```

```
        }
    }
    flow[source][sink] = flow[sink][source] = 0;
    mat[source][sink] = sk;
    mat[sink][source] = ks;
    maxFlow(n, mat, sink, source, flow);
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            mat[i][j] += bf[i][j];
            flow[i][j] += bf[i][j];
        }
    }
    for (j = i = 0; i < n; j += flow[source][i++]);
    return j;
}
```

最大流无流量(邻接阵)

```
//求网络最大流,邻接阵形式
//返回最大流量
//传入网络节点数 n,容量 mat,源点 source,汇点 sink
//注意 mat 矩阵被修改
//注意，只是为了省那么点空间，速度反而比有流量的慢!

const int MAXN = 100;
const int INF = 1000000000;

int maxFlow(int n, int mat[][MAXN], int source, int sink) {
    int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;
    while (true) {
        for (i = 0; i < n; i++) {
            v[i] = c[i] = 0;
        }
        for (c[source] = INF;;) {
            for (j = -1, i = 0; i < n; i++) {
                if (!v[i] && c[i] && (j == -1 || c[i] > c[j])) {
                    j = i;
                }
            }
            if (j < 0) {
                return ret;
            }
            if (j == sink) {
                break;
            }
            for (v[j] = 1, i = 0; i < n; i++) {
                if (mat[j][i] > c[i] && c[j] > c[i]) {
                    c[i] = mat[j][i] < c[j] ? mat[j][i] : c[j];
```

```
                    p[i] = j;
                }
            }
        }
        for (ret += j = c[i = sink]; i != source; i = p[i]) {
            mat[p[i]][i] -= j;
            mat[i][p[i]] += j;
        }
    }
}
```

最小费用最大流(邻接阵)

```
//求网络最小费用最大流,邻接阵形式
//返回最大流量,flow 返回每条边的流量,netcost 返回总费用
//传入网络节点数 n,容量 mat,单位费用 cost,源点 source,汇点 sink

const int MAXN = 100;
const int INF = 1000000000;

int minCostMaxFlow(int n, const int mat[][MAXN], int cost[][MAXN], int source, int
sink, int flow[][MAXN], int & netcost) {
    int pre[MAXN], min[MAXN], d[MAXN], i, j, t, tag;
    if (source == sink) {
        return INF;
    } for (i = 0; i < n; i++) {
        for (j = 0; j < n; flow[i][j++] = 0);
    }
    for (netcost = 0;;) {
        for (i = 0; i < n; i++) {
            pre[i] = 0;
            min[i] = INF;
        }
        pre[source] = source + 1;
        min[source] = 0;
        d[source] = INF;
        for (tag = 1; tag;) {
            for (tag = t = 0; t < n; t++) {
                if (d[t]) {
                    for (i = 0; i < n; i++) {
                        if ((j = mat[t][i] - flow[t][i]) && min[t] + cost[t][i] < m
in[i]) {
                            tag = 1;
                            min[i] = min[t] + cost[t][i];
                            pre[i] = t + 1;
                            d[i] = d[t] < j ? d[t] : j;
                        } else if ((j = flow[i][t]) && min[t] < INF && min[t] - cos
t[i][t] < min[i]) {
```

```
                            tag = 1;
                            min[i] = min[t] - cost[i][t];
                            pre[i] = -t - 1;
                            d[i] = d[t] < j ? d[t] : j;
                        }
                    }
                }
            }
        }
        if (!pre[sink]) {
            break;
        }
        for (netcost += min[sink] * d[i = sink]; i != source;) {
            if (pre[i] > 0) {
                flow[pre[i] - 1][i] += d[sink];
                i = pre[i] - 1;
            } else {
                flow[i][-pre[i] - 1] -= d[sink];
                i = -pre[i] - 1;
            }
        }
    }
    for (j = i = 0; i < n; j += flow[source][i++]);
    return j;
}
```

最短路最大流(邻接阵)

```
//求网络最大流,邻接阵形式, 最坏情况复杂度 O(EV^2)
//返回最大流量,f 返回每条边的流量,返回最大流量
//传入网络节点数 n,容量 mat,源点 s,汇点 t

const int MAXN = 210;
const int INF = 1000000000;

int maxFlow(int n, const int mat[][MAXN], int s, int t, int f[][MAXN]) {
    int v[MAXN], pre[MAXN], dis[MAXN], pos[MAXN], d[MAXN], i, j, k, ret;
    if (s == t) {
        return INF;
    }
    for (i = 0; i < n; i++) {
        for (dis[i] = INF, v[i] = pos[i] = j = 0; j < n; f[i][j++] = 0);
    }
    for (dis[t] = 0, i = 0; i < n; i++) {
        for (j = 0, k = -1; j < n; j++) {
            if (!v[j] && (k < 0 || dis[j] < dis[k])) {
                k = j;
            }
```

```
        }
        for (v[k] = 1, j = 0; j < n; j++) {
            if (!v[j] && mat[j][k] && dis[k] < dis[j]) {
                dis[j] = dis[k] + 1;
            }
        }
    }
    for (d[i = s] = INF, pre[s] = -1; dis[s] < n;) {
        if (i == t) {
            for (; i != s; i = j) {
                f[i][j = pre[i]] = -(f[pre[i]][i] += d[t]);
            }
        } else if (pos[i] >= n) {
            for (j = 0, k = INF; j < n; j++) {
                if (mat[i][j] > f[i][j] && dis[j] < k) {
                    k = dis[j];
                    pos[i] = j;
                }
            }
            dis[i] = k + 1;
            i = pre[i];
        } else if ((k = mat[i][j = pos[i]] - f[i][j]) && dis[i] == dis[j] + 1 && j
!= pre[i]) {
            d[j] = d[i] < k ? d[i]: k;
            pre[j] = i;
            i = j;
        } else {
            pos[i]++;
        }
    }
    for (i = ret = 0; i < n; i++) {
        ret += f[s][i];
    }
    return ret;
}
```

最短路最大流(邻接表形式,邻接阵接口)

```
//求网络最大流,邻接表形式, 邻接阵接口, 最坏情况复杂度O(EV^2)
//返回最大流量,f返回每条边的流量,返回最大流量
//传入网络节点数n,容量mat,源点s,汇点t

const int MAXN = 210;
const int INF = 1000000000;

int maxFlow(int n, const int mat[][MAXN], int s, int t, int f[][MAXN]) {
    int g[MAXN][MAXN], v[MAXN], pre[MAXN], dis[MAXN], pos[MAXN], d[MAXN], i, j, k,
ret, next;
```

```
    if (s == t) {
        return INF;
    }
    for (i = 0; i < n; i++) {
        dis[i] = INF;
        pos[i] = 1;
        for (g[i][0] = v[i] = j = 0; j < n; f[i][j++] = 0) {
            if (mat[i][j] || mat[j][i]) {
                g[i][++g[i][0]] = j;
            }
        }
    }
    for (dis[t] = 0, i = 0; i < n; i++) {
        for (j = 0, k = -1; j < n; j++) {
            if (!v[j] && (k < 0 || dis[j] < dis[k])) {
                k = j;
            }
        }
        for (v[k] = 1, j = 0; j < n; j++) {
            if (!v[j] && mat[j][k] && dis[k] < dis[j]) {
                dis[j] = dis[k] + 1;
            }
        }
    }
    for (d[i = s] = INF, pre[s] = -1; dis[s] < n;) {
        if (i == t) {
            for (; i != s; i = j) {
                f[i][j = pre[i]] = -(f[pre[i]][i] += d[t]);
            }
        } else if (pos[i] > g[i][0]) {
            for (j = 1, k = INF; j <= g[i][0]; j++) {
                if (mat[i][next = g[i][j]] > f[i][next] && dis[next] < k) {
                    k = dis[next];
                    pos[i] = j;
                }
            }
            dis[i] = k + 1;
            i = pre[i];
        } else if ((k = mat[i][j = g[i][pos[i]]] - f[i][j]) && dis[i] == dis[j] + 1
) {
            d[j] = d[i] < k ? d[i]: k;
            pre[j] = i;
            i = j;
        } else {
            pos[i]++;
        }
    }
```

```
    for (i = ret = 0; i < n; i++) {
        ret += f[s][i];
    }
    return ret;
}
```

先流推进

```
//邻接阵形式
//返回最大流量,f 返回每条边的流量,返回最大流量
//传入网络节点数 n,容量 mat,源点 s,汇点 t
#include <list>
using namespace std;

const int MAXN = 210;
const int INF = 1000000000;

int liftToFront(int n, const int mat[][MAXN], int s, int t, int f[][MAXN]) {
    if (s == t) {
        return INF;
    }
    int h[MAXN], e[MAXN], pos[MAXN], k, ret = 0;
    list<int> que;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            f[i][j] = 0;
        }
    }
    for (int i = 0; i < n; ++i) {
        h[i] = e[i] = pos[i] = 0;
    }
    h[s] = n;
    for (int i = 0; i < n; ++i) {
        if (mat[s][i]) {
            f[i][s] = -(f[s][i] = e[i] = mat[s][i]);
        }
    }
    for (int i = 0; i < n; ++i) {
        if (i != s && i != t) {
            que.push_back(i);
        }
    }
    for (list <int> ::iterator it = que.begin(); it != que.end(); ++it) {
        int old = h[k = *it];
        for (int i = pos[k]; e[k] > 0; i = pos[k]) {
            if (i >= n) {
                int height = INF;
                for (int j = 0; j < n; ++j) {
```

```
                    if (mat[k][j] > f[k][j] && h[j] < height) {
                        height = h[j];
                    }
                }
                h[k] = height + 1;
                pos[k] = 0;
            } else if (mat[k][i] > f[k][i] && h[k] == h[i] + 1) {
                int d = e[k] < (mat[k][i] - f[k][i]) ? e[k]: (mat[k][i] - f[k][i]);
                f[i][k] =  - (f[k][i] += d);
                e[k] -= d;
                e[i] += d;
            } else {
                ++pos[k];
            }
        }
        if (h[k] > old) {
            que.erase(it), que.push_front(k);
            it = que.begin();
        }
    }
    for (int i = 0; i < n; ++i) {
        ret += f[s][i];
    }
    return ret;
}
```

最高标号先流推进

```
//邻接表形式，邻接阵接口，复杂度O(V^3)
//返回最大流量,f返回每条边的流量,返回最大流量
//传入网络节点数n,容量mat,源点s,汇点t
const int MAXN = 210;
const int INF = 1000000000;

int maxFlow(int n, const int mat[][MAXN], int s, int t, int f[][MAXN]) {
    int g[MAXN][MAXN], cur[MAXN], h[MAXN], e[MAXN], q[MAXN], l[MAXN * 2][MAXN], i,
j, k, head, tail, ret, checked, p, o;
    for (i = 0; i < n; i++) {
        h[i] =  - 1;
        cur[i] = 1;
        for (e[i] = g[i][0] = j = 0; j < n; f[i][j++] = 0) {
            if (mat[i][j] || mat[j][i]) {
                g[i][++g[i][0]] = j;
            }
        }
    }
    for (i = 0; i < 2 * n; i++) {
        l[i][0] = 0;
```

```
    }
    for (l[h[q[head = 0] = t] = 0][++l[0][0]] = t, tail = 1; head < tail; head++) {
        for (i = 1; i <= g[j = q[head]][0]; i++) {
            if (h[k = g[j][i]] < 0) {
                h[k] = h[j] + 1;
                q[tail++] = k;
                if (k != s) {
                    l[h[k]][++l[h[k]][0]] = k;
                }
            }
        }
    }
    for (h[s] = n, i = 1; i <= g[s][0]; i++) {
        j = g[s][i];
        f[j][s] = -(f[s][j] = e[j] = mat[s][j]);
    }
    for (i = n; i; i--) {
        for (checked = 0, j = l[i][0]; j;) {
            if ((k = l[i][j]) == s) {
                j--;
            } else if (!e[k]) {          //Full
                if (checked) {           //Update
                    if (i && l[i][0] == 1) {
                        for (p = h[k] + 1; p < n; l[n][0] += l[p][0], l[p++][0] = 0
) {
                            for (o = 1; o <= l[p][0]; o++) {
                                l[n][++l[n][0]] = l[p][o];
                                h[l[p][o]] = n;
                            }
                        }
                    }
                    l[h[k]][++l[h[k]][0]] = k;
                    l[i][j] = l[i][l[i][0]--];
                    i = h[k];
                    break;
                } else {
                    j--;
                }
            } else if (cur[k] > g[k][0]) {     //Relabel
                for (checked = p = 1, o = INF; p <= g[k][0]; p++) {
                    if (mat[k][g[k][p]] > f[k][g[k][p]] && h[g[k][p]] < o) {
                        o = h[g[k][p]];
                    }
                }
                h[k] = o + 1, cur[k] = 1;
            } else if ((o = mat[k][p = g[k][cur[k]]] - f[k][p]) && h[k] == h[p] + 1
) {
```

```
                o = o < e[k] ? o : e[k];
                f[p][k] = -(f[k][p] += o);
                e[k] -= o;
                e[p] += o;
            } else {
                cur[k]++;
            }
        }
    }
    for (ret = 0, i = 1; i <= g[s][0]; i++) {
        ret += f[s][g[s][i]];
    }
    return ret;
}
```

图论－应用

欧拉回路(邻接阵)

```
//求欧拉回路或欧拉路,邻接阵形式,复杂度O(n^2)
//返回路径长度,path返回路径(有向图时得到的是反向路径)
//传入图的大小n和邻接阵mat,不相邻点边权0
//可以有自环与重边,分为无向图和有向图

const int MAXN = 100;

void findPathU(int n, int mat[][MAXN], int now, int & step, int * path) {
    int i;
    for (i = n - 1; i >= 0; i--) {
        while (mat[now][i]) {
            mat[now][i]--;
            mat[i][now]--;
            findPathU(n, mat, i, step, path);
        }
    }
    path[step++] = now;
}

void findPathD(int n, int mat[][MAXN], int now, int & step, int * path) {
    int i;
    for (i = n - 1; i >= 0; i--) {
        while (mat[now][i]) {
            mat[now][i]--;
            findPathD(n, mat, i, step, path);
        }
    }
    path[step++] = now;
}

int euclidPath(int n, int mat[][MAXN], int start, int * path) {
    int ret = 0;
    findPathU(n, mat, start, ret, path);
//    findPathD(n, mat, start, ret, path);
    return ret;
}
```

树的前序表转化

```
//将用边表示的树转化为前序表示的树
//传入节点数n和邻接表list[],邻接表必须是双向的,会在函数中释放
//pre[]返回前序表,map[]返回前序表中的节点到原来节点的映射
const int MAXN = 10000;
```

```
struct Node {
    int to;
    Node * next;
};

void prenode(int n, Node * list[], int * pre, int * map, int * v, int now, int last
, int & id) {
    Node * t;
    int p = id++;
    for (v[map[p] = now] = 1, pre[p] = last; list[now];) {
        t = list[now];
        list[now] = t->next;
        if (!v[t->to]) {
            prenode(n, list, pre, map, v, t->to, p, id);
        }
    }
}

void makepre(int n, Node * list[], int * pre, int * map) {
    int v[MAXN], id = 0, i;
    for (i = 0; i < n; v[i++] = 0);
    prenode(n, list, pre, map, v, 0,  - 1, id);
}
```

树的优化算法

```
const int MAXN = 1000;

//最大顶点独立集
int maxNodeIndependent(int n, int * pre, int * set) {
    int c[MAXN], i, ret = 0;
    for (i = 0; i < n; i++) {
        c[i] = set[i] = 0;
    }
    for (i = n - 1; i >= 0; i--) {
        if (!c[i]) {
            set[i] = 1;
            if (pre[i] != -1) {
                c[pre[i]] = 1;
            }
            ret++;
        }
    }
    return ret;
}

//最大边独立集
int maxEdgeIndependent(int n, int * pre, int * set) {
```

```
    int c[MAXN], i, ret = 0;
    for (i = 0; i < n; i++) {
        c[i] = set[i] = 0;
    }
    for (i = n - 1; i >= 0; i--) {
        if (!c[i] && pre[i] != -1 && !c[pre[i]]) {
            set[i] = 1;
            c[pre[i]] = 1;
            ret++;
        }
    }
    return ret;
}

//最小顶点覆盖集
int minNodeCover(int n, int * pre, int * set) {
    int c[MAXN], i, ret = 0;
    for (i = 0; i < n; i++) {
        c[i] = set[i] = 0;
    }
    for (i = n - 1; i >= 0; i--) {
        if (!c[i] && pre[i] != -1 && !c[pre[i]]) {
            set[i] = 1;
            c[pre[i]] = 1;
            ret++;
        }
    }
    return ret;
}

//最小顶点支配集
int minNodeDominant(int n, int * pre, int * set) {
    int c[MAXN], i, ret = 0;
    for (i = 0; i < n; i++) {
        c[i] = set[i] = 0;
    }
    for (i = n - 1; i >= 0; i--) {
        if (!c[i] && (pre[i] == -1 || !set[pre[i]])) {
            if (pre[i] !=  - 1) {
                set[pre[i]] = 1;
                c[pre[i]] = 1;
                if (pre[pre[i]] != -1) {
                    c[pre[pre[i]]] = 1;
                }
            } else {
                set[i] = 1;
            }
```

```
            ret++;
        }
    }
    return ret;
}
```

拓扑排序(邻接阵)

```
//拓扑排序,邻接阵形式,复杂度O(n^2)
//如果无法完成排序,返回0,否则返回1,ret返回有序点列
//传入图的大小n和邻接阵mat,不相邻点边权0
const int MAXN = 100;

bool toposort(int n, int mat[][MAXN], int * ret) {
    int d[MAXN], i, j, k;
    for (i = 0; i < n; i++) {
        for (d[i] = j = 0; j < n; d[i] += mat[j++][i]);
    }
    for (k = 0; k < n; ret[k++] = i) {
        for (i = 0; d[i] && i < n; i++);
        if (i == n) {
            return false;
        }
        for (d[i] = -1, j = 0; j < n; j++) {
            d[j] -= mat[i][j];
        }
    }
    return true;
}
```

最佳边割集

```
//最佳边割集
const int MAXN = 100;
const int INF = 1000000000;

int maxFlow(int n, int mat[][MAXN], int source, int sink) {
    int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;
    while (true) {
        for (i = 0; i < n; i++) {
            v[i] = c[i] = 0;
        }
        for (c[source] = INF;;) {
            for (j = -1, i = 0; i < n; i++) {
                if (!v[i] && c[i] && (j == -1 || c[i] > c[j])) {
                    j = i;
                }
            }
            if (j < 0) {
```

```
                return ret;
            }
            if (j == sink) {
                break;
            }
            for (v[j] = 1, i = 0; i < n; i++) {
                if (mat[j][i] > c[i] && c[j] > c[i]) {
                    c[i] = mat[j][i] < c[j] ? mat[j][i]: c[j];
                    p[i] = j;
                }
            }
        }
        for (ret += j = c[i = sink]; i != source; i = p[i]) {
            mat[p[i]][i] -= j;
            mat[i][p[i]] += j;
        }
    }
}

int bestEdgeCut(int n, int mat[][MAXN], int source, int sink, int set[][2], int & m
incost) {
    int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, l, ret = 0, last;
    if (source == sink) {
        return -1;
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            m0[i][j] = mat[i][j];
        }
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            m[i][j] = m0[i][j];
        }
    }
    mincost = last = maxFlow(n, m, source, sink);
    for (k = 0; k < n && last; k++) {
        for (l = 0; l < n && last; l++) {
            if (m0[k][l]) {
                for (i = 0; i < n + n; i++) {
                    for (j = 0; j < n + n; j++) {
                        m[i][j] = m0[i][j];
                    }
                }
                m[k][l] = 0;
                if (maxFlow(n, m, source, sink) == last - mat[k][l]) {
                    set[ret][0] = k;
```

```
                    set[ret++][1] = l;
                    m0[k][l] = 0;
                    last -= mat[k][l];
                }
            }
        }
    }
    return ret;
}
```

最佳点割集

```
//最佳顶点割集
const int MAXN = 100;
const int INF = 1000000000;

int maxFlow(int n, int mat[][MAXN], int source, int sink) {
    int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;
    while (true) {
        for (i = 0; i < n; i++) {
            v[i] = c[i] = 0;
        }
        for (c[source] = INF;;) {
            for (j = -1, i = 0; i < n; i++) {
                if (!v[i] && c[i] && (j == -1 || c[i] > c[j])) {
                    j = i;
                }
            }
            if (j < 0) {
                return ret;
            }
            if (j == sink) {
                break;
            }
            for (v[j] = 1, i = 0; i < n; i++) {
                if (mat[j][i] > c[i] && c[j] > c[i]) {
                    c[i] = mat[j][i] < c[j] ? mat[j][i] : c[j];
                    p[i] = j;
                }
            }
        }
        for (ret += j = c[i = sink]; i != source; i = p[i]) {
            mat[p[i]][i] -= j;
            mat[i][p[i]] += j;
        }
    }
}
```

```
int bestVertexCut(int n, int mat[][MAXN], int * cost, int source, int sink, int * s
et, int & mincost) {
    int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, ret = 0, last;
    if (source == sink || mat[source][sink]) {
        return -1;
    }
    for (i = 0; i < n + n; i++) {
        for (j = 0; j < n + n; j++) {
            m0[i][j] = 0;
        }
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (mat[i][j]) {
                m0[i][n + j] = INF;
            }
        }
    }
    for (i = 0; i < n; i++) {
        m0[n + i][i] = cost[i];
    }
    for (i = 0; i < n + n; i++) {
        for (j = 0; j < n + n; j++) {
            m[i][j] = m0[i][j];
        }
    }
    mincost = last = maxFlow(n + n, m, source, n + sink);
    for (k = 0; k < n && last; k++) {
        if (k != source && k != sink) {
            for (i = 0; i < n + n; i++) {
                for (j = 0; j < n + n; j++) {
                    m[i][j] = m0[i][j];
                }
            }
            m[n + k][k] = 0;
            if (maxFlow(n + n, m, source, n + sink) == last - cost[k]) {
                set[ret++] = k;
                m0[n + k][k] = 0;
                last -= cost[k];
            }
        }
    }
    return ret;
}
```

最小边割集

```
//最小边割集
```

```
const int MAXN = 100;
const int INF = 1000000000;

int maxFlow(int n, int mat[][MAXN], int source, int sink) {
    int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;
    while (true) {
        for (i = 0; i < n; i++) {
            v[i] = c[i] = 0;
        }
        for (c[source] = INF;;) {
            for (j = -1, i = 0; i < n; i++) {
                if (!v[i] && c[i] && (j == -1 || c[i] > c[j])) {
                    j = i;
                }
            }
            if (j < 0) {
                return ret;
            } if (j == sink) {
                break;
            } for (v[j] = 1, i = 0; i < n; i++) {
                if (mat[j][i] > c[i] && c[j] > c[i]) {
                    c[i] = mat[j][i] < c[j] ? mat[j][i] : c[j];
                    p[i] = j;
                }
            }
        }
        for (ret += j = c[i = sink]; i != source; i = p[i]) {
            mat[p[i]][i] -= j;
            mat[i][p[i]] += j;
        }
    }
}

int minEdgeCut(int n, int mat[][MAXN], int source, int sink, int set[][2]) {
    int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, l, ret = 0, last;
    if (source == sink) {
        return  - 1;
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            m0[i][j] = (mat[i][j] != 0);
        }
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            m[i][j] = m0[i][j];
        }
```

```
    }
    last = maxFlow(n, m, source, sink);
    for (k = 0; k < n && last; k++) {
        for (l = 0; l < n && last; l++) {
            if (m0[k][l]) {
                for (i = 0; i < n + n; i++) {
                    for (j = 0; j < n + n; j++) {
                        m[i][j] = m0[i][j];
                    }
                }
                m[k][l] = 0;
                if (maxFlow(n, m, source, sink) < last) {
                    set[ret][0] = k;
                    set[ret++][1] = l;
                    m0[k][l] = 0;
                    last--;
                }
            }
        }
    }
    return ret;
}
```

最小点割集

```
//最小顶点割集
const int MAXN = 100;
const int INF = 1000000000;

int maxFlow(int n, int mat[][MAXN], int source, int sink) {
    int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;
    while (true) {
        for (i = 0; i < n; i++) {
            v[i] = c[i] = 0;
        }
        for (c[source] = INF;;) {
            for (j = -1, i = 0; i < n; i++) {
                if (!v[i] && c[i] && (j == -1 || c[i] > c[j])) {
                    j = i;
                }
            }
            if (j < 0) {
                return ret;
            }
            if (j == sink) {
                break;
            }
            for (v[j] = 1, i = 0; i < n; i++) {
```

```
                if (mat[j][i] > c[i] && c[j] > c[i]) {
                    c[i] = mat[j][i] < c[j] ? mat[j][i] : c[j];
                    p[i] = j;
                }
            }
        }
        for (ret += j = c[i = sink]; i != source; i = p[i]) {
            mat[p[i]][i] -= j;
            mat[i][p[i]] += j;
        }
    }
}

int minVertexCut(int n, int mat[][MAXN], int source, int sink, int * set) {
    int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, ret = 0, last;
    if (source == sink || mat[source][sink]) {
        return -1;
    }
    for (i = 0; i < n + n; i++) {
        for (j = 0; j < n + n; j++) {
            m0[i][j] = 0;
        }
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (mat[i][j]) {
                m0[i][n + j] = INF;
            }
        }
    }
    for (i = 0; i < n; i++) {
        m0[n + i][i] = 1;
    }
    for (i = 0; i < n + n; i++) {
        for (j = 0; j < n + n; j++) {
            m[i][j] = m0[i][j];
        }
    }
    last = maxFlow(n + n, m, source, n + sink);
    for (k = 0; k < n && last; k++) {
        if (k != source && k != sink) {
            for (i = 0; i < n + n; i++) {
                for (j = 0; j < n + n; j++) {
                    m[i][j] = m0[i][j];
                }
            }
            m[n + k][k] = 0;
```

```
            if (maxFlow(n + n, m, source, n + sink) < last) {
                set[ret++] = k;
                m0[n + k][k] = 0;
                last--;
            }
        }
    }
    return ret;
}
```

最小路径覆盖

```
//最小路径覆盖,O(n^3)
//求解最小的路径覆盖图中所有点,有向图无向图均适用
//注意此问题等价二分图最大匹配,可以用邻接表或正向表减小复杂度
//返回最小路径条数,pre 返回前指针(起点-1),next 返回后指针(终点-1)
#include <cstring>
const int MAXN = 310;
#define _clr(x) memset(x, 0xff, sizeof(int) * n)

int hungary(int n, const bool mat[][MAXN], int * match1, int * match2) {
    int s[MAXN], t[MAXN], p, q, ret = 0, i, j, k;
    _clr(match1);
    _clr(match2);
    for (i = 0; i < n; ret += (match1[i++] >= 0)) {
        _clr(t);
        for (s[p = q = 0] = i; p <= q && match1[i] < 0; p++) {
            for (k = s[p], j = 0; j < n && match1[i] < 0; j++) {
                if (mat[k][j] && t[j] < 0) {
                    s[++q] = match2[j];
                    t[j] = k;
                    if (s[q] < 0) {
                        for (p = j; p >= 0; j = p) {
                            match2[j] = k = t[j];
                            p = match1[k];
                            match1[k] = j;
                        }
                    }
                }
            }
        }
    }
    return ret;
}

inline int pathCover(int n, const bool mat[][MAXN], int * pre, int * next) {
    return n - hungary(n, mat, next, pre);
}
```

图论－生成树

最小生成树(kruskal 邻接表)

```
//无向图最小生成树,kruskal 算法,邻接表形式,复杂度 O(mlogm)
//返回最小生成树的长度,传入图的大小 n 和邻接表 list
//可更改边权的类型,edge[][2]返回树的构造,用边集表示
//如果图不连通,则对各连通分支构造最小生成树,返回总长度
#include <cstring>
const int MAXN = 200;
const int INF = 1000000000;

#define _run(x) for(; p[t = x]; x = p[x], p[t] = (p[x] ? p[x] : x))
#define _run_both _run(i); _run(j)

class DSet {
public:
    int p[MAXN], t;
    void init() {
        memset(p, 0, sizeof(p));
    }
    void setFriend(int i, int j) {
        _run_both;
        p[i] = (i == j ? 0 : j);
    }
    bool isFriend(int i, int j) {
        _run_both;
        return i == j && i;
    }
};

typedef double elemType;

struct Edge {
    int from, to;
    elemType len;
    Edge * next;
};

struct HeapNode {
    int a, b;
    elemType len;
};

#define _cp(a,b) ((a).len < (b).len)
```

```
class MinHeap {
public:
    HeapNode h[MAXN * MAXN];
    int n, p, c;
    void init() {
        n = 0;
    }
    void ins(HeapNode e) {
        for (p = ++n; p > 1 && _cp(e, h[p >> 1]); h[p] = h[p >> 1], p >>= 1);
        h[p] = e;
    }
    bool del(HeapNode & e) {
        if (!n) {
            return false;
        }
        e = h[p = 1];
        for (c = 2; c < n && _cp(h[c += (c < n - 1 && _cp(h[c + 1], h[c]))], h[n]);
 c <<= 1) {
            h[p] = h[c];
            p = c;
        }
        h[p] = h[n--];
        return true;
    }
};

elemType kruskal(int n, const Edge * list[], int edge[][2]) {
    DSet u;
    MinHeap h;
    const Edge * t;
    HeapNode e;
    elemType ret = 0;
    int i, m = 0;
    u.init(), h.init();
    for (i = 0; i < n; i++) {
        for (t = list[i]; t; t = t->next) {
            if (i < t->to) {
                e.a = i;
                e.b = t->to;
                e.len = t->len;
                h.ins(e);
            }
        }
    }
    while (m < n - 1 && h.del(e)) {
        if (!u.isFriend(e.a + 1, e.b + 1)) {
            edge[m][0] = e.a;
```

```
            edge[m][1] = e.b;
            ret += e.len;
            u.setFriend(e.a + 1, e.b + 1);
        }
    }
    return ret;
}
```

最小生成树(prim+priority_queue 邻接表)

```
// 不是太苛刻的情况下推荐使用
// 复杂度为O(|E|+|V|lg|V|)，但常数较大，而且复杂度不是很严格
// 边权非负!

#define Rec pair<T, int>

template<class T>
T Prim(int n, vector<pair<int, T> > e[], T mind[], int* pre = NULL)
{
    int s = 0;
    priority_queue<Rec, vector<Rec>, greater<Rec> > q;
    vector<bool> mark(n, false);

    // pre为NULL则不做记录
    if (pre != NULL) {
        fill(pre, pre + n, -1);
        pre[s] = s;
    }
    // mind初始化部分注意修改
    fill(mind, mind + n, numeric_limits<T>::max());
    mind[s] = T();
    q.push(make_pair(T(), s));

    T ret = T();

    while (!q.empty()) {
        s = q.top().second;
        if (!mark[s]) {
            mark[s] = true;
            ret += q.top().first;
            q.pop();
            for (typename vector<pair<int, T> >::const_iterator i = e[s].begin(); i
 != e[s].end(); ++i) {
                if (!mark[i->first] && mind[i->first] > i->second) {
                    mind[i->first] = i->second;
                    if (pre != NULL) {
                        pre[i->first] = s;
                    }
```

```
                    q.push(make_pair(mind[i->first], i->first));
                }
            }
        }
        else {
            q.pop();
        }
    }

    return ret;
}
```

最小生成树(prim+binary_heap 邻接表)

```
//无向图最小生成树,prim 算法+二分堆,邻接表形式,复杂度 O(mlogm)
//返回最小生成树的长度,传入图的大小 n 和邻接表 list
//可更改边权的类型,pre[]返回树的构造,用父结点表示,根节点(第一个)pre 值为-1
//必须保证图的连通的!
const int MAXN = 200;
const int INF = 1000000000;

typedef double elemType;

struct Edge {
    int from, to;
    elemType len;
    Edge * next;
};

#define _cp(a, b) ((a).d < (b).d)

struct HeapNode {
    elemType d;
    int v;
};

class Heap {
public:
    HeapNode h[MAXN * MAXN];
    int n, p, c;
    void init() {
        n = 0;
    }
    void ins(HeapNode e) {
        for (p = ++n; p > 1 && _cp(e, h[p >> 1]); h[p] = h[p >> 1], p >>= 1);
        h[p] = e;
    }
    bool del(HeapNode & e) {
```

```
        if (!n) {
            return false;
        }
        e = h[p = 1];
        for (c = 2; c < n && _cp(h[c += (c < n - 1 && _cp(h[c + 1], h[c]))], h[n]);
 c <<= 1) {
            h[p] = h[c];
            p = c;
        }
        h[p] = h[n--];
        return true;
    }
};

elemType prim(int n, const Edge * list[], int * pre) {
    Heap h;
    elemType mind[MAXN], ret = 0;
    const Edge * t;
    HeapNode e;
    int v[MAXN], i;
    for (i = 0; i < n; i++) {
        mind[i] = INF;
        v[i] = 0;
        pre[i] = -1;
    }
    h.init();
    e.v = 0;
    e.d = 0;
    h.ins(e);
    while (h.del(e)) {
        if (!v[e.v]) {
            v[e.v] = 1;
            ret += e.d;
            for (t = list[e.v]; t; t = t->next) {
                if (!v[t->to] && t->len < mind[t->to]) {
                    pre[t->to] = t->from;
                    mind[e.v = t->to] = e.d = t->len;
                    h.ins(e);
                }
            }
        }
    }
    return ret;
}
```

最小生成树(prim+mapped_heap 邻接表)

```
//无向图最小生成树,prim 算法+映射二分堆,邻接表形式,复杂度 O(mlogn)
```

```
//返回最小生成树的长度,传入图的大小 n 和邻接表 list
//可更改边权的类型,pre[]返回树的构造,用父结点表示,根节点(第一个)pre 值为-1
//必须保证图的连通的!
const int MAXN = 200;
const int INF = 1000000000;

typedef double elemType;

struct Edge {
    int from, to;
    elemType len;
    Edge * next;
};

#define _cp(a, b) ((a) < (b))

struct Heap {
    elemType h[MAXN + 1];
    int ind[MAXN + 1], map[MAXN + 1], n, p, c;
    void init() {
        n = 0;
    }
    void ins(int i, elemType e) {
        for (p = ++n; p > 1 && _cp(e, h[p >> 1]); p >>= 1) {
            h[map[ind[p] = ind[p >> 1]] = p] = h[p >> 1];
        }
        h[map[ind[p] = i] = p] = e;
    }
    bool del(int i, elemType & e) {
        i = map[i];
        if (i < 1 || i > n) {
            return false;
        }
        for (e = h[p = i]; p > 1; p >>= 1) {
            h[map[ind[p] = ind[p >> 1]] = p] = h[p >> 1];
        }
        for (c = 2; c < n && _cp(h[c += (c < n - 1 && _cp(h[c + 1], h[c]))], h[n]);
 c <<= 1) {
            h[map[ind[p] = ind[c]] = p] = h[c];
            p = c;
        }
        h[map[ind[p] = ind[n]] = p] = h[n];
        n--;
        return true;
    }
    bool delmin(int & i, elemType & e) {
        if (n < 1) {
```

```
            return false;
        }
        i = ind[1];
        e = h[p = 1];
        for (c = 2; c < n && _cp(h[c += (c < n - 1 && _cp(h[c + 1], h[c]))], h[n]);
 c <<= 1) {
            h[map[ind[p] = ind[c]] = p] = h[c];
            p = c;
        }
        h[map[ind[p] = ind[n]] = p] = h[n];
        n--;
        return true;
    }
};

elemType prim(int n, const Edge * list[], int * pre) {
    Heap h;
    elemType mind[MAXN], ret = 0, e;
    const Edge * t;
    int v[MAXN], i;
    for (h.init(), i = 0; i < n; i++) {
        mind[i] = (i ? INF : 0);
        v[i] = 0;
        pre[i] = -1;
        h.ins(i, mind[i]);
    }
    while (h.delmin(i, e)) {
        v[i] = 1;
        ret += e;
        for (t = list[i]; t; t = t->next) {
            if (!v[t->to] && t->len < mind[t->to]) {
                pre[t->to] = t->from;
                h.del(t->to, e);
                h.ins(t->to, mind[t->to] = t->len);
            }
        }
    }
    return ret;
}
```

最小生成树(prim 邻接阵)

```
//无向图最小生成树,prim 算法,邻接阵形式,复杂度 O(n^2)
//返回最小生成树的长度,传入图的大小 n 和邻接阵 mat,不相邻点边权 INF
//可更改边权的类型,pre[]返回树的构造,用父结点表示,根节点(第一个)pre 值为-1
//必须保证图的连通的!
const int MAXN = 200;
const int INF = 1000000000;
```

```
template <class elemType>
elemType prim(int n, const elemType mat[][MAXN], int * pre) {
    elemType mind[MAXN], ret = 0;
    int v[MAXN], i, j, k;
    for (i = 0; i<n; i++) {
        mind[i] = INF;
        v[i] = 0;
        pre[i] = -1;
    }
    for (mind[j = 0] = 0; j<n; j++) {
        for (k = -1, i = 0; i<n; i++) {
            if (!v[i] && (k == -1 || mind[i]<mind[k])) {
                k = i;
            }
        }
        v[k] = 1;
        ret += mind[k];
        for (i = 0; i < n; i++) {
            if (!v[i] && mat[k][i] < mind[i]) {
                mind[i] = mat[pre[i] = k][i];
            }
        }
    }
    return ret;
}
```

最小树形图(邻接阵)

```
//多源最小树形图,edmonds 算法,邻接阵形式,复杂度 O(n^3)
//返回最小生成树的长度,构造失败返回负值
//传入图的大小 n 和邻接阵 mat,不相邻点边权 INF
//可更改边权的类型,pre[]返回树的构造,用父结点表示
//传入时 pre[]数组清零,用-1 标出源点
#include <cstring>
const int MAXN = 120;
const int INF = 1000000000;

template <class elemType> elemType edmonds(int n, const elemType mat[][MAXN * 2], i
nt * pre) {
    elemType ret = 0;
    int c[MAXN * 2][MAXN * 2], L[MAXN * 2], p[MAXN * 2], m = n, t, i, j, k;
    for (i = 0; i < n; L[i] = i, i++);
    do {
        memset(c, 0, sizeof(c));
        memset(p, 0xff, sizeof(p));
        for (t = m, i = 0; i < m; c[i][i] = 1, i++);
        for (i = 0; i < t; i++) {
```

```
            if (L[i] == i && pre[i] != -1) {
                for (j = 0; j < m; j++) {
                    if (L[j] == j && i != j && mat[j][i] < INF && (p[i] == -1 || ma
t[j][i] < mat[p[i]][i])) {
                        p[i] = j;
                    }
                }
                if ((pre[i] = p[i]) == -1) {
                    return -1;
                }
                if (c[i][p[i]]) {
                    for (j = 0; j <= m; mat[j][m] = mat[m][j] = INF, j++);
                    for (k = i; L[k] != m; L[k] = m, k = p[k]) {
                        for (j = 0; j < m; j++) {
                            if (L[j] == j) {
                                if (mat[j][k] - mat[p[k]][k] < mat[j][m]) {
                                    mat[j][m] = mat[j][k] - mat[p[k]][k];
                                }
                                if (mat[k][j] < mat[m][j]) {
                                    mat[m][j] = mat[k][j];
                                }
                            }
                        }
                    }
                    c[m][m] = 1, L[m] = m, m++;
                }
                for (j = 0; j < m; j++) {
                    if (c[i][j]) {
                        for (k = p[i]; k != -1 && L[k] == k; c[k][j] = 1, k = p[k])
;
                    }
                }
            }
        }
    } while (t < m);
    for (; m-- > n; pre[k] = pre[m]) {
        for (i = 0; i < m; i++) {
            if (L[i] == m) {
                for (j = 0; j < m; j++) {
                    if (pre[j] == m && mat[i][j] == mat[m][j]) {
                        pre[j] = i;
                    }
                }
                if (mat[pre[m]][m] == mat[pre[m]][i] - mat[pre[i]][i]) {
                    k = i;
                }
            }
```

```
        }
    }
    for (i = 0; i < n; i++) {
        if (pre[i] != -1) {
            ret += mat[pre[i]][i];
        }
    }
    return ret;
}

// 最小树形图，就是给有向带权图中指定一个特殊的点 v，求一棵有向生成树 T，使得该有向树的根为 v，并且 T 中所有边的总权值最小。
```

次最小生成树

```
// 次小生成树，复杂度 O(n^2)
// 传入邻接阵 mat，不存在边权 inf
// 返回次小生成树长度和树的构造 pre[]
// 如返回 inf 则不存在次小生成树
// 必须保证图的连通
const int maxn = 100;
const int inf = 1000000000;
typedef int elem_t;

elem_t prim(int n, elem_t mat[][maxn], int pre) {
    elem_t min[maxn], ret = 0;
    int v[maxn], i, j, k;

    for (i = 0; i < n; ++i)
        min[i] = inf, v[i] = 0, pre[i] = -1;
    for (min[j = 0] = 0; j < n; ++j) {
        for (k = -1, i = 0; i < n; ++i) if (!v[i] && (k == -1 || min[i] < min[k]))
            k = i;
        for (v[k] = 1, ret += min[k], i = 0; i < n; ++i)
            if (!v[i] && mat[k][i] < min[i])
                min[i] = mat[pre[i] = k][i];
    }

    return ret;
}

elem_t sbmst(int n, elem_t mat[][maxn], int pre) {
    elem_t min = inf, t, ret = prim(n, mat, pre);
    int i, j, ti, tj;
    for (i = 0; i < n; ++i) for (j = 0; j < n && pre[i] != -1; ++j) if (i != j && p
re[i] != j && pre[j] != i)
        if (mat[j][i] < inf && (t = mat[j][i] - mat[pre[i]][i]) < min)
            min = t, ti = i, tj = j;
```

```
    pre[ti] = tj;
    return ret + min;
}
```

图论－最短路径

最短路径(单源 SPFA)

```
// Shortest Path Faster Algorithm
// 可以处理负环，若有负环则返回 fasle，比 bellmanFord 效率高，建议替代 bellmanFord 使用
// 为了效率可以不使用 STL
template<class T>
bool SPFA(int n, int s, vector<pair<int, T> > e[], T mind[], int* pre = NULL)
{
    queue<int> q;
    vector<int> d(n, 0);
    vector<bool> mark(n, false);

    // pre 为 NULL 则不做记录
    if (pre != NULL) {
        fill(pre, pre + n, -1);
        pre[s] = s;
    }
    // mind 初始化部分注意修改
    fill(mind, mind + n, numeric_limits<T>::max());
    mind[s] = 0;
    mark[s] = true;
    q.push(s);
    while (!q.empty()) {
        s = q.front();
        mark[s] = false;
        q.pop();
        for (typename vector<pair<int, T> >::const_iterator i =e[s].begin(); i != e
[s].end(); ++i) {
            if (mind[i->first] > mind[s] + i->second) {
                mind[i->first] = mind[s] + i->second;
                d[i->first] = d[s] + 1;
                if (pre != NULL) {
                    pre[i->first] = s;
                }
                if (d[i->first] >= n) {
                    return false;
                }
                if (!mark[i->first]) {
                    mark[i->first] = true;
                    q.push(i->first);
                }
            }
        }
    }
```

```
    return true;
}
```

最短路径(单源 bellman_ford 邻接阵)

```
//单源最短路径,bellman_ford 算法,邻接阵形式,复杂度 O(n^3)
//求出源 s 到所有点的最短路经,传入图的大小 n 和邻接阵 mat
//返回到各点最短距离 mind[]和路径 pre[],pre[i]记录 s 到 i 路径上 i 的父结点,pre[s]=-1
//可更改路权类型,路权可为负,若图包含负环则求解失败,返回 0
//优化:先删去负边使用 dijkstra 求出上界,加速迭代过程
const int MAXN = 200;
const int INF = 1000000000;

template <class elemType>
bool bellmanFord(int n, const elemType mat[][MAXN], int s, elemType * mind, int * p
re) {
    int v[MAXN], i, j, k, tag;
    for (i = 0; i < n; i++) {
        mind[i] = INF;
        v[i] = 0;
        pre[i] = -1;
    }
    for (mind[s] = 0, j = 0; j < n; j++) {
        for (k = -1, i = 0; i < n; i++) {
            if (!v[i] && (k == -1 || mind[i] < mind[k])) {
                k = i;
            }
        }
        for (v[k] = 1, i = 0; i < n; i++) {
            if (!v[i] && mat[k][i] >= 0 && mind[k] + mat[k][i] < mind[i]) {
                mind[i] = mind[k] + mat[pre[i] = k][i];
            }
        }
    }
    for (tag = 1, j = 0; tag && j <= n; j++) {
        for (tag = i = 0; i < n; i++) {
            for (k = 0; k < n; k++) {
                if (mind[k] + mat[k][i] < mind[i]) {
                    mind[i] = mind[k] + mat[pre[i] = k][i];
                    tag = 1;
                }
            }
        }
    }
    return j <= n;
}
```

最短路径(单源 bellman_ford 邻接表)

```
//单源最短路径,bellman_ford 算法,邻接表形式,复杂度 O(nm)
//求出源 s 到所有点的最短路经,传入图的大小 n 和邻接表 list
//返回到各点最短距离 mind[]和路径 pre[],pre[i]记录 s 到 i 路径上 i 的父结点,pre[s]=-1
//可更改路权类型,路权可为负,若图包含负环则求解失败,返回 0
//优化:先删去负边使用 dijkstra 求出上界,加速迭代过程
const int MAXN = 200;
const int INF = 1000000000;
typedef int elemType;
struct Edge {
    int from, to;
    elemType len;
    Edge * next;
};

bool bellmanFord(int n, const Edge * list[], int s, elemType * mind, int * pre) {
    int v[MAXN], i, j, k, tag;
    const Edge * t;
    elemType len;
    for (i = 0; i < n; i++) {
        mind[i] = INF;
        v[i] = 0;
        pre[i] = -1;
    }
    for (mind[s] = 0, j = 0; j < n; j++) {
        for (k = -1, i = 0; i < n; i++) {
            if (!v[i] && (k == -1 || mind[i] < mind[k])) {
                k = i;
            }
        }
        for (v[k] = 1, t = list[k]; t; t = t->next) {
            if (!v[i = t->to] && (len = t->len) >= 0 && mind[k] + len < mind[i]) {
                mind[i] = mind[pre[i] = k] + len;
            }
        }
    }
    for (tag = 1, j = 0; tag && j <= n; j++) {
        for (tag = k = 0; k < n; k++) {
            for (t = list[k]; t; t = t->next) {
                if (mind[k] + (len = t->len) < mind[i = t->to]) {
                    mind[i] = mind[pre[i] = k] + len;
                    tag = 1;
                }
            }
        }
    }
```

```
    return j <= n;
}



#include <vector>
using namespace std;

const int MAXN = 200;
const int INF = 1000000000;

template <class elemType>
bool bellmanFord(int n, const vector<pair<int, elemType> > list[], int s, elemType
* mind, int * pre) {
    int v[MAXN], i, j, k, t, tag;
    elemType len;
    for (i = 0; i<n; i++) {
        mind[i] = INF;
        v[i] = 0;
        pre[i] = -1;
    }
    for (mind[s] = 0, j = 0; j<n; j++) {
        for (k = -1, i = 0; i<n; i++) {
            if (!v[i] && (k == -1 || mind[i]<mind[k])) {
                k = i;
            }
        }
        for (v[k] = 1, i = 0; i < list[k].size(); i++) {
            if (!v[t = list[k][i].first] && (len = list[k][i].second) >= 0 && mind[
k] + len < mind[t]) {
                mind[t] = mind[pre[t] = k] + len;
            }
        }
    }
    for (tag = 1, j = 0; tag && j <= n; j++) {
        for (tag = k = 0; k < n; k++) {
            for (i = 0; i < list[k].size(); i++) {
                if (mind[k] + (len = list[k][i].second) < mind[t = list[k][i].first
]) {
                    mind[t] = mind[pre[t] = k] + len;
                    tag = 1;
                }
            }
        }
    }
    return j <= n;
}
```

最短路径(单源 dijkstra+bfs 邻接表)

```
//单源最短路径,用于路权相等的情况,dijkstra 优化为 bfs,邻接表形式,复杂度 O(m)
//求出源 s 到所有点的最短路经,传入图的大小 n 和邻接表 list,边权值 len
//返回到各点最短距离 mind[]和路径 pre[],pre[i]记录 s 到 i 路径上 i 的父结点,pre[s]=-1
//可更改路权类型,但必须非负且相等!
const int MAXN = 200;
const int INF = 1000000000;

struct Edge {
    int from, to;
    Edge * next;
};

template <class elemType> void dijkstra(int n, const Edge * list[], elemType len, i
nt s, elemType * mind, int * pre) {
    Edge * t;
    int i, que[MAXN], f = 0, r = 0, p = 1, L = 1;
    for (i = 0; i<n; i++) {
        mind[i] = INF;
    }
    mind[que[0] = s] = 0;
    pre[s] = -1;
    for (; r <= f; L++, r = f + 1, f = p - 1) {
        for (i = r; i <= f; i++) {
            for (t = list[que[i]]; t; t = t->next) {
                if (mind[t->to] == INF) {
                    mind[que[p++] = t->to] = len * L;
                    pre[t->to] = que[i];
                }
            }
        }
    }
}
```

最短路径(单源 dijkstra+priority_queue 邻接表)

```
// 不是太苛刻的情况下推荐使用
// 复杂度为 O(|E|+|V|lg|E|)，但常数较大，而且复杂度不是很严格
// 边权非负!

#define Rec pair<T, int>

template<class T>
void Dijkstra(int n, int s, vector<pair<int, T> > e[], T mind[], int* pre = NULL)
{
    priority_queue<Rec, vector<Rec>, greater<Rec> > q;
    vector<bool> mark(n, false);
```

```
    // pre 为 NULL 则不做记录
    if (pre != NULL) {
        fill(pre, pre + n, -1);
        pre[s] = s;
    }
    // mind 初始化部分注意修改
    fill(mind, mind + n, numeric_limits<T>::max());
    mind[s] = T();
    q.push(make_pair(T(), s));
    while (!q.empty()) {
        s = q.top().second;
        q.pop();
        if (mark[s]) {
            continue;
        }
        else {
            mark[s] = true;
        }
        mark[s] = true;
        for (typename vector<pair<int, T> >::const_iterator i = e[s].begin(); i !=
e[s].end(); ++i) {
            if (!mark[i->first] && mind[i->first] > mind[s] + i->second) {
                mind[i->first] = mind[s] + i->second;
                if (pre != NULL) {
                    pre[i->first] = s;
                }
                q.push(make_pair(mind[i->first], i->first));
            }
        }
    }
}
```

最短路径(单源 dijkstra+binary_heap 邻接表)

```
//在时间常数不是很苛刻的情况下建议使用 Dijkstra+priority_queue

//单源最短路径,dijkstra 算法+二分堆,邻接表形式,复杂度 O(mlogm)
//求出源 s 到所有点的最短路经,传入图的大小 n 和邻接表 list
//返回到各点最短距离 mind[]和路径 pre[],pre[i]记录 s 到 i 路径上 i 的父结点,pre[s]=-1
//可更改路权类型,但必须非负!
const int MAXN = 200;
const int INF = 1000000000;

typedef int elemType;

struct Edge {
    int from, to;
```

```
    elemType len;
    Edge * next;
};

#define _cp(a, b) ((a).d < (b).d)

struct HeapNode {
    elemType d;
    int v;
};

class Heap {
public:
    HeapNode h[MAXN * MAXN];
    int n, p, c;
    void init() {
        n = 0;
    }
    void ins(HeapNode e) {
        for (p = ++n; p > 1 && _cp(e, h[p >> 1]); h[p] = h[p >> 1], p >>= 1);
        h[p] = e;
    }
    bool del(HeapNode & e) {
        if (!n) {
            return false;
        }
        e = h[p = 1];
        for (c = 2; c < n && _cp(h[c += (c < n - 1 && _cp(h[c + 1], h[c]))], h[n]);
 c <<= 1) {
            h[p] = h[c];
            p = c;
        }
        h[p] = h[n--];
        return true;
    }
};

void dijkstra(int n, const Edge * list[], int s, elemType * mind, int * pre) {
    Heap h;
    const Edge * t;
    HeapNode e;
    int v[MAXN], i;
    for (i = 0; i < n; i++) {
        mind[i] = INF;
        v[i] = 0;
        pre[i] = -1;
    }
```

```
    h.init();
    mind[e.v = s] = e.d = 0;
    h.ins(e);
    while (h.del(e)) {
        if (!v[e.v]) {
            for (v[e.v] = 1, t = list[e.v]; t; t = t->next) {
                if (!v[t->to] && mind[t->from] + t->len < mind[t->to]) {
                    pre[t->to] = t->from;
                    mind[e.v = t->to] = e.d = mind[t->from] + t->len;
                    h.ins(e);
                }
            }
        }
    }
}
```

最短路径(单源 dijkstra+mapped_heap 邻接表)

```
//在时间常数不是很苛刻的情况下建议使用 Dijkstra+priority_queue

//单源最短路径,dijkstra 算法+映射二分堆,邻接表形式,复杂度 O(mlogn)
//求出源 s 到所有点的最短路经,传入图的大小 n 和邻接表 list
//返回到各点最短距离 mind[]和路径 pre[],pre[i]记录 s 到 i 路径上 i 的父结点,pre[s]=-1
//可更改路权类型,但必须非负!
const int MAXN = 200;
const int INF = 1000000000;
typedef int elemType;

struct Edge {
    int from, to;
    elemType len;
    Edge * next;
};

#define _cp(a, b) ((a) < (b))

class Heap {
public:
    elemType h[MAXN + 1];
    int ind[MAXN + 1], map[MAXN + 1], n, p, c;
    void init() {
        n = 0;
    }
    void ins(int i, elemType e) {
        for (p = ++n; p > 1 && _cp(e, h[p >> 1]); p >>= 1) {
            h[map[ind[p] = ind[p >> 1]] = p] = h[p >> 1];
        }
        h[map[ind[p] = i] = p] = e;
```

```
    }
    bool del(int i, elemType & e) {
        i = map[i];
        if (i < 1 || i > n) {
            return false;
        }
        for (e = h[p = i]; p > 1; p >>= 1) {
            h[map[ind[p] = ind[p >> 1]] = p] = h[p >> 1];
        }
        for (c = 2; c < n && _cp(h[c += (c < n - 1 && _cp(h[c + 1], h[c]))], h[n]);
 c <<= 1) {
            h[map[ind[p] = ind[c]] = p] = h[c];
            p = c;
        }
        h[map[ind[p] = ind[n]] = p] = h[n];
        n--;
        return true;
    }
    bool delmin(int & i, elemType & e) {
        if (n < 1) {
            return false;
        }
        i = ind[1];
        e = h[p = 1];
        for (c = 2; c < n && _cp(h[c += (c < n - 1 && _cp(h[c + 1], h[c]))], h[n]);
 c <<= 1) {
            h[map[ind[p] = ind[c]] = p] = h[c];
            p = c;
        }
        h[map[ind[p] = ind[n]] = p] = h[n];
        n--;
        return true;
    }
};

void dijkstra(int n, Edge * list[], int s, elemType * mind, int * pre) {
    Heap h;
    Edge * t;
    elemType e;
    int v[MAXN], i;
    for (h.init(), i = 0; i < n; i++) {
        mind[i] = ((i == s) ? 0 : INF);
        v[i] = 0;
        pre[i] = -1;
        h.ins(i, mind[i]);
    }
    while (h.delmin(i, e)) {
```

```
        for (v[i] = 1, t = list[i]; t; t = t->next) {
            if (!v[t->to] && mind[i] + t->len < mind[t->to]) {
                pre[t->to] = i;
                h.del(t->to, e);
                mind[t->to] = e = mind[i] + t->len;
                h.ins(t->to, e);
            }
        }
    }
}
```

最短路径(单源 dijkstra 邻接阵)

```
//单源最短路径,dijkstra 算法,邻接阵形式,复杂度 O(n^2)
//求出源 s 到所有点的最短路经,传入图的顶点数 n,(有向)邻接矩阵 mat
//返回到各点最短距离 mind[]和路径 pre[],pre[i]记录 s 到 i 路径上 i 的父结点,pre[s]=-1
//可更改路权类型,但必须非负!
const int MAXN = 200;
const int INF = 1000000000;

template <class elemType>
void dijkstra(int n, const elemType mat[][MAXN], int s, elemType * mind, int * pre)
 {
    int v[MAXN], i, j, k;
    for (i = 0; i < n; i++) {
        mind[i] = INF;
        v[i] = 0;
        pre[i] = -1;
    }
    for (mind[s] = 0, j = 0; j < n; j++) {
        for (k = -1, i = 0; i < n; i++) {
            if (!v[i] && (k == -1 || mind[i] < mind[k])) {
                k = i;
            }
        }
        for (v[k] = 1, i = 0; i < n; i++) {
            if (!v[i] && mind[k] + mat[k][i] < mind[i]) {
                mind[i] = mind[k] + mat[pre[i] = k][i];
            }
        }
    }
}
```

最短路径(多源 floyd_warshall 邻接阵)

```
//多源最短路径,floyd_warshall 算法,复杂度 O(n^3)
//求出所有点对之间的最短路经,传入图的大小和邻接阵
//返回各点间最短距离 mind[]和路径 pre[],pre[i][j]记录 i 到 j 最短路径上 j 的父结点
//可更改路权类型,路权必须非负!
```

```
const int MAXN = 200;
const int INF = 1000000000;

template <class elemType>
void floydWarshall(int n, const elemType mat[][MAXN], elemType mind[][MAXN], int pr
e[][MAXN]) {
    int i, j, k;
    for (i = 0; i<n; i++) {
        for (j = 0; j<n; j++) {
            mind[i][j] = mat[i][j];
            pre[i][j] = (i == j) ? -1: i;
        }
    }
    for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                if (mind[i][k] + mind[k][j] < mind[i][j]) {
                    mind[i][j] = mind[i][k] + mind[k][j];
                    pre[i][j] = pre[k][j];
                }
            }
        }
    }
}
```

应用

Joseph 问题

```
// Joseph's Problem
// input: n,m        -- the number of persons, the interval between persons
// output:        -- return the reference of last person

int josephus0(int n, int m) {
    if (n == 2) {
        return (m % 2) ? 2 : 1;
    }
    int v = (m + josephus0(n - 1, m)) % n;
    if (v == 0) {
        v = n;
    }
    return v;
}

int josephus(int n, int m) {
    if (m == 1) {
        return n;
    } else if (n == 1) {
        return 1;
    } else if (m >= n) {
        return josephus0(n, m);
    }
    int l = (n / m) * m;
    int j = josephus(n - (n / m), m);
    if (j <= n - l) {
        return l + j;
    }
    j -= n - l;
    int t = (j / (m - 1)) * m;
    if ((j % (m - 1)) == 0) {
        return t - 1;
    }
    return t + (j % (m - 1));
}

//Joseph 问题模拟
//传入 n,m, r 返回出环的序列
//时间复杂度 O(nlogn)
#include <cstring>

const int MAXN = 32768;
```

```
void josephus(int n, int m, int r[]) {
    int d[MAXN * 2], i, j, nbase, p, t;
    for (nbase = 1; nbase < n; nbase <<= 1);
    memset(d, 0, sizeof(d));
    for (i = 0; i < n; i++) {
        d[nbase + i] = 1;
    }
    for (i = nbase - 1; i; i--) {
        d[i] = d[i << 1] + d[i << 1 | 1];
    }
    for (j = i = 0; i < n; i++) {
        for (j = t = (j - 1+m) % (n - i), p = 1; p < nbase;) {
            d[p]--;
            if (t < d[p << 1]) {
                p = p << 1;
            } else {
                t -= d[p << 1];
                p = p << 1 | 1;
            }
        }
        r[i] = p - nbase;
        d[p]--;
    }
}
```

布尔母函数

```
//布尔母函数
//判m[]个价值为w[]的货币能否构成value
//适合m[]较大w[]较小的情况
//返回布尔量
//传入货币种数n,个数m[],价值w[]和目标值value
const int MAXV = 100000;

bool genfunc(int n, const int * m, const int * w, int value) {
    int i, j, k, c;
    char r[MAXV];
    for (r[0] = i = 1; i <= value; r[i++] = 0);
    for (i = 0; i < n; i++) {
        for (j = 0; j < w[i]; j++) {
            c = m[i] * r[k = j];
            while ((k += w[i]) <= value) {
                if (r[k]) {
                    c = m[i];
                } else if (c) {
                    r[k] = 1;
                    c--;
```

```
                }
            }
            if (r[value]) {
                return true;
            }
        }
    }
    return false;
}
```

模式匹配(kmp)

```
//模式匹配,kmp 算法,复杂度 O(m+n)
//返回匹配位置,-1 表示匹配失败,传入匹配串和模式串和长度
//可更改元素类型,更换匹配函数
const int MAXN = 10000;
#define _match(a, b) ((a) == (b))

template <class elemType>
int patMatch(int ls, const elemType * str, int lp, const elemType * pat) {
    int fail[MAXN] = {-1}, i = 0, j;
    for (j = 1; j < lp; j++) {
        for (i = fail[j - 1]; i >= 0 && !_match(pat[i + 1], pat[j]); i = fail[i]);
        fail[j] = (_match(pat[i + 1], pat[j]) ? i + 1 :  - 1);
    }
    for (i = j = 0; i < ls && j < lp; i++) {
        if (_match(str[i], pat[j])) {
            j++;
        } else if (j) {
            j = fail[j - 1] + 1;
            i--;
        }
    }
    return j == lp ? (i - lp) : -1;
}

// 统计次数

#define MAXN 10000
#define _match(a,b) ((a)==(b))
typedef char elem_t;

int pat_match(int ls,elem_t* str,int lp,elem_t* pat){
    int ret = 0;
    int fail[MAXN]={-1},i=0,j;
    for (j=1;j<lp;j++){
        for (i=fail[j-1];i>=0&&!_match(pat[i+1],pat[j]);i=fail[i]);
        fail[j]=(_match(pat[i+1],pat[j])?i+1:-1);
```

```
    }
    for (i=j=0;i<ls;i++) {
        if (_match(str[i],pat[j])) {
            j++;
            if (j == lp) {
                ++ret;
                j=fail[j-1]+1;
            }
        }
        else if (j)
            j=fail[j-1]+1,i--;
    }
    return ret;
}
```

逆序对数

```
//序列逆序对数,复杂度O(nlogn)
//传入序列长度和内容,返回逆序对数
//可更改元素类型和比较函数
#include <cstring>
const int MAXN = 1000000;
#define _cp(a,b) ((a) <= (b))

typedef int elemType;
elemType tmp[MAXN];

long long inv(int n, elemType * a) {
    int left = n >> 1, r = n - left, i, j;
    long long ret = (r > 1 ? (inv(left, a) + inv(r, a + left)) : 0);
    for (i = j = 0; i <= left; tmp[i + j] = a[i], i++) {
        for (ret += j; j < r && (i == left || !_cp(a[i], a[left + j])); tmp[i + j]
= a[left + j], j++);
    }
    memcpy(a, tmp, sizeof(elemType) * n);
    return ret;
}
```

字符串最小表示

```
/*
    求字符串的最小表示
    输入：字符串
    返回：字符串最小表示的首字母位置(0...size-1)
*/
#include <vector>
using namespace std;

template <class T>
```

```
int minString(const vector<T>  & str) {
    int i, j, k;
    vector<T> ss(str.size() << 1);
    for (i = 0; i<str.size(); i++) {
        ss[i] = ss[i + str.size()] = str[i];
    }
    for (i = k = 0, j = 1; k<str.size() && i<str.size() && j<str.size();) {
        for (k = 0; k<str.size() && ss[i + k] == ss[j + k]; k++);
        if (k<str.size()) {
            if (ss[i + k]> ss[j + k]) {
                i += k + 1;
            } else {
                j += k + 1;
            }
            if (i == j) {
                j++;
            }
        }
    }
    return i < j ? i : j;
}
```

线性规划

```
#include <iostream>
#include <cmath>
using namespace std;

const double EPS = 1e-9;
const int MAX_N = 100;
const int MAX_M = 100;
typedef double LPMAT[MAX_M + 1][MAX_M + MAX_N + 1];
typedef double LPRET[MAX_N + 1];

inline bool d_zero(double x) {
    return fabs(x) < EPS;
};

inline bool d_less(double x, double y) {
    return x + EPS < y;
};

/** 线性规划.
 * 求 b1 x1 + b2 x2 + ... + bn xn 在 x1, x2, ..., xn >= 0 时的最大值
 * m 为限制条件方程数, n 为变量数目
 * 限制条件为
 *  a_i1 x1 + a_i2 x2 + ... + a_in xn <= ci (1 <= i <= m, ci >= 0 (!!!))
 * mat 传入系数矩阵
```

```
 * mat[0] 表示目标方程, 即 mat[0][1..n] 为 b1 .. bn
 * mat[1..m]表示限制方程, mat[1..m][0] 表示 c1, c2, ... cm
 * 其余 mat[i][j] 表示 a[i][j]
 * 注意函数会改变 mat 的值
 *   i \ j 0    1    2   .... n
 *   0     0   b1   b2   .... bn
 *   1     c1  a11  a12  .... a1n
 *   ..............................
 *   m     cm  am1  am2  .... amn
 * 找到解返回 true, ans 返回最大值, ret[1..n] 分别返回 x1..xn 的取值
 * 如果不存在最大值返回 false
 * 不可能无解, 因为 x1 = x2 = ... = xn = 0 必为一组解
 */
bool lp(int m, int n, LPMAT mat, double& ans, LPRET ret) {
    static int p[MAX_M + 1], q[MAX_M + MAX_N + 1];
    static double trial[MAX_M + MAX_N + 1];
    int i, j, k, l, s, h;
    double z, zbuf;

    mat[0][0] = 0;
    for (i = 0; i <= m; i++) {
        for (j = n; j > 0; j--)
            mat[i][j + m] = (i == 0) ? -mat[i][j] : mat[i][j];

        for (j = m; j > 0; j--)
            mat[i][j] = (i == j) ? 1 : 0;

        p[i] = q[i] = i;
    }

    bool flag = true;
    while (flag) {
        flag = false;
        for (j = m + n; j > 0; j--) {
            if (!d_less(mat[0][j], 0))
                continue;
            for (i = 1, l = 0; i <= m; i++) {
                if (!d_less(0, mat[i][j]))
                    continue;
                if (l == 0) {
                    l = i, s = 0;
                } else {
                    for (h = 0; ; h++) {
                        if (h == s)
                            trial[s++] = mat[l][h] / mat[l][j];
                        z = mat[i][h] / mat[i][j];
                        if (trial[h] != z)
```

```
                            break;
                    }
                    if (d_less(z, trial[h]))
                        l = i, trial[h] = z, s = h + 1;
                }
            }
            if (l == 0)
                // The maximum is infinite
                return false;

            for (k = 0, z = mat[l][j]; k <= m + n; k++)
                if (!d_zero(mat[l][k]))
                    mat[l][k] = mat[l][k] / z;

            for (i = 0; i <= m; i++) {
                if (i == l)
                    continue;
                for (k = 0, z = mat[i][j]; k <= m + n; k++)
                    mat[i][k] = (k == j || d_zero(zbuf = mat[i][k] - z * mat[l][k])
) ?
                                0 : zbuf;
            }
            q[p[l]] = 0, p[l] = j, q[j] = l;
            flag = true;
            break;
        };
    };
    ans = mat[0][0];

    for (i = 1, j = m + 1; j <= m + n; i++, j++)
        ret[i] = (q[j]) ? mat[q[j]][0] : 0;

    /* 此处可用来计算 (u1, u2, ..., um) 其中 ui >= 0, 使得在
     * a1j u1 + a2j u2 + ... + amj um >= bj (1 <= j <= n)
     * 的限制条件下 c1 u1 + c2 u2 + ... + cm um 最大.
     * 当函数返回 false 的时候此处无解. */
    //for (j = 1; j <= m; j++)
    //    u[i] = mat[0][j];

    return true;
}
```

最长公共单调子序列

```
/**
 * 最长公共递增子序列, 时间复杂度O(n^2 * logn), 空间O(n^2)
 * n为a的大小, m为b的大小
 * 结果在ans中
```

```
 * "define _cp(a,b) ((a)<(b))"求解最长严格递增序列
 */
const int MAXN = 1000;
#define _cp(a,b) ((a) < (b))

typedef int elemType;

elemType DP[MAXN][MAXN];
int num[MAXN], p[1 << 20];
int lis(int n, const elemType * a, int m, const elemType * b, elemType * ans) {
    int i, j, l, r, k;
    DP[0][0] = 0;
    num[0] = (b[0] == a[0]);
    for (i = 1; i < m; i++) {
        num[i] = (b[i] == a[0]) || num[i - 1];
        DP[i][0] = 0;
    }
    for (i = 1; i < n; i++) {
        if (b[0] == a[i] && !num[0]) {
            num[0] = 1;
            DP[0][0] = i << 10;
        }
        for (j = 1; j < m; j++) {
            for (k = ((l = 0) + (r = num[j - 1] - 1)) >> 1; l <= r; k = (l + r) >>
1) {
                if (_cp(a[DP[j - 1][k] >> 10], a[i])) {
                    l = k + 1;
                } else {
                    r = k - 1;
                }
            }
            if (l < num[j - 1] && i == (DP[j - 1][l] >> 10)) {
                if (l >= num[j]) {
                    DP[j][num[j]++] = DP[j - 1][l];
                } else {
                    DP[j][l] = _cp(a[DP[j][l] >> 10], a[i]) ? DP[j][l] : DP[j - 1][
l];

                }
            }
            if (b[j] == a[i]) {
                for (k = ((l = 0) + (r = num[j] - 1)) >> 1; l <= r; k = (l + r) >>
1) {
                    if (_cp(a[DP[j][k] >> 10], a[i])) {
                        l = k + 1;
                    } else {
                        r = k - 1;
                    }
```

```
                }
                DP[j][l] = (i << 10) + j;
                num[j] += (l >= num[j]);
                p[DP[j][l]] = l ? DP[j][l - 1]:  - 1;
            }
        }
    }

    for (k = DP[m - 1][i = num[m - 1] - 1]; i >= 0; k = p[k]) {
        ans[i--] = a[k >> 10];
    }
    return num[m - 1];
}
```

最长子序列

```
//最长单调子序列,复杂度O(nlogn)
//注意最小序列覆盖和最长序列的对应关系,例如
//"define _cp(a,b) ((a)>(b))"求解最长严格递减序列,则
//"define _cp(a,b) (!((a)>(b)))"求解最小严格递减序列覆盖
//可更改元素类型和比较函数
const int MAXN = 10000;
#define _cp(a,b) ((a)>(b))

template <class elemType>
int subseq(int n, const elemType * a) {
    int b[MAXN + 1], i, l, r, m, ret = 0;
    for (i = 0; i < n; b[l] = i++, ret += (l> ret)) {
        for (m = ((l = 1) + (r = ret)) >> 1; l <= r; m = (l + r) >> 1) {
            if (_cp(a[b[m]], a[i])) {
                l = m + 1;
            } else {
                r = m - 1;
            }
        }
    }
    return ret;
}

template <class elemType>
int subseq(int n, const elemType * a, elemType * ans) {
    int b[MAXN + 1], p[MAXN], i, l, r, m, ret = 0;
    for (i = 0; i < n; p[b[l] = i++] = b[l - 1], ret += (l> ret)) {
        for (m = ((l = 1) + (r = ret)) >> 1; l <= r; m = (l + r) >> 1) {
            if (_cp(a[b[m]], a[i])) {
                l = m + 1;
            } else {
                r = m - 1;
```

```
            }
        }
    }
    for (m = b[i = ret]; i; ans[--i] = a[m], m = p[m]);
    return ret;
}
```

最大子阵和

```
//求最大子阵和,复杂度O(n^3)
//传入阵的大小m,n和内容mat[][]
//返回最大子阵和,重载返回子阵位置(maxsum=list[s1][s2]+...+list[e1][e2])
//可更改元素类型
const int MAXN = 100;

template <class elemType>
elemType maxsum(int m, int n, const elemType mat[][MAXN]) {
    elemType matsum[MAXN][MAXN + 1], ret, sum;
    int i, j, k;
    for (i = 0; i < m; i++) {
        for (matsum[i][j = 0] = 0; j < n; j++) {
            matsum[i][j + 1] = matsum[i][j] + mat[i][j];
        }
    }
    for (ret = mat[0][j = 0]; j < n; j++) {
        for (k = j; k < n; k++) {
            for (sum = 0, i = 0; i < m; i++) {
                sum = (sum > 0 ? sum : 0) + matsum[i][k + 1] - matsum[i][j], ret =
(sum > ret ? sum : ret);
            }
        }
    }
    return ret;
}

template <class elemType>
elemType maxsum(int m, int n, const elemType mat[][MAXN], int & s1, int & s2, int &
 e1, int & e2) {
    elemType matsum[MAXN][MAXN + 1], ret, sum;
    int i, j, k, s;
    for (i = 0; i < m; i++) {
        for (matsum[i][j = 0] = 0; j < n; j++) {
            matsum[i][j + 1] = matsum[i][j] + mat[i][j];
        }
    }
    for (ret = mat[s1 = e1 = 0][s2 = e2 = j = 0]; j < n; j++) {
        for (k = j; k < n; k++) {
            for (sum = 0, s = i = 0; i < m; i++, s = (sum > 0 ? s : i)) {
```

```
                if ((sum = (sum > 0 ? sum : 0) + matsum[i][k + 1] - matsum[i][j]) >
 ret) {
                    ret = sum;
                    s1 = s;
                    s2 = i;
                    e1 = j;
                    e2 = k;
                }
            }
        }
    }
    return ret;
}
```

后缀数组

```
//Linear Suffix Array

//Function suffixArray is copy from
//Linear Work Suffix Array Construction
//Written by Juha K arkk ainen , Peter Sandersy AND Stefan Burkhardt

//Function Height is written by myself

//Count the number of the same substring in two given string
//the pairs of the same substrings are not shorter than N
#include<cstdio>
#include<cstring>

inline bool leq(int a1, int a2, int b1, int b2){ // lexicographic order
    return(a1 < b1 || a1 == b1 && a2 <= b2);
} // for pairs

inline bool leq(int a1, int a2, int a3, int b1, int b2, int b3){
    return(a1 < b1 || a1 == b1 && leq(a2,a3, b2,b3));
} // and triples
// stably sort a[0..n-1] to b[0..n-1] with keys in 0..K from r

void radixPass(int* a, int* b, int* r, int n, int K){ // count occurrences
    int* c = new int[K + 1]; // counter array
    for (int i = 0; i <= K; i++) c[i] = 0; // reset counters
    for (int i = 0; i < n; i++) c[r[a[i]]]++; // count occurrences
    for (int i = 0, sum = 0; i <= K; i++){ // exclusive prefix sums
        int t = c[i]; c[i] = sum; sum += t;
    }
    for (int i = 0; i < n; i++) b[c[r[a[i]]]++] = a[i]; // sort
    delete [] c;
}
```

```
// find the suffix array SA of T[0..n-1] in {1..K}^n
// require T[n]=T[n+1]=T[n+2]=0, n>=2
void suffixArray(int* T, int* SA, int n, int K) {
    int n0=(n+2)/3, n1=(n+1)/3, n2=n/3, n02=n0+n2;
    int* R = new int[n02 + 3]; R[n02]= R[n02+1]= R[n02+2]=0;
    int* SA12 = new int[n02 + 3]; SA12[n02]=SA12[n02+1]=SA12[n02+2]=0;
    int* R0 = new int[n0];
    int* SA0 = new int[n0];
    //******* Step 0: Construct sample ********
    // generate positions of mod 1 and mod 2 suffixes
    // the "+(n0-n1)" adds a dummy mod 1 suffix if n%3 == 1
    for (int i=0, j=0; i < n+(n0-n1); i++) if (i%3 != 0) R[j++] = i;
    //******* Step 1: Sort sample suffixes ********
    // lsb radix sort the mod 1 and mod 2 triples
    radixPass(R , SA12, T+2, n02, K);
    radixPass(SA12, R , T+1, n02, K);
    radixPass(R , SA12, T , n02, K);
    // find lexicographic names of triples and
    // write them to correct places in R
    int name = 0, c0 = -1, c1 = -1, c2 = -1;
    for (int i = 0; i < n02; i++) {
        if (T[SA12[i]] != c0 || T[SA12[i]+1] != c1 || T[SA12[i]+2] != c2){
            name++; c0 = T[SA12[i]]; c1 = T[SA12[i]+1]; c2 = T[SA12[i]+2];
        }
        if (SA12[i] % 3 == 1) { R[SA12[i]/3] = name; } // write to R1
        else { R[SA12[i]/3 + n0] = name; } // write to R2
    }
    // recurse if names are not yet unique
    if (name < n02) {
        suffixArray(R, SA12, n02, name);
        // store unique names in R using the suffix array
        for (int i = 0; i < n02; i++) R[SA12[i]] = i + 1;
    } else // generate the suffix array of R directly
        for (int i = 0; i < n02; i++) SA12[R[i] - 1] = i;
        //******* Step 2: Sort nonsample suffixes ********
        // stably sort the mod 0 suffixes from SA12 by their first character
    for (int i=0, j=0; i < n02; i++) if (SA12[i] < n0) R0[j++] = 3*SA12[i];
    radixPass(R0, SA0, T, n0, K);
    //******* Step 3: Merge ********
    // merge sorted SA0 suffixes and sorted SA12 suffixes
    for (int p=0, t=n0-n1, k=0; k < n; k++) {
        #define GetI() (SA12[t] < n0 ? SA12[t] * 3 + 1 : (SA12[t] - n0) * 3 + 2)
        int i = GetI(); // pos of current offset 12 suffix
        int j = SA0[p]; // pos of current offset 0 suffix
        if (SA12[t] < n0 ? // different compares for mod 1 and mod 2 suffixes
            leq(T[i], R[SA12[t] + n0], T[j], R[j/3]) :
```

```
            leq(T[i],T[i+1],R[SA12[t]-n0+1], T[j],T[j+1],R[j/3+n0])){ // suffix fro
m SA12 is smaller
                SA[k] = i; t++;
                if (t == n02) // done --- only SA0 suffixes left
                    for (k++; p < n0; p++, k++) SA[k] = SA0[p];
                } else { // suffix from SA0 is smaller
                    SA[k] = j; p++;
                    if (p == n0) // done --- only SA12 suffixes left
                        for (k++; t < n02; t++, k++) SA[k] = GetI();
                }
    }
    delete [] R; delete [] SA12; delete [] SA0; delete [] R0;
}

// LCP(i,j)=min{hgt[k]|i+1<=k<=j}
// hgt[k]=LCP(i-1,i)
// O(4n)
void Height(int* T,int* SA,int* height,int n){
    int* rank=new int[n+3];
    int* h=new int[n+3];
    for(int i=0;i<n;i++){
        rank[SA[i]]=i;
    }
    for(int i=0;i<n;i++){
        if(rank[i]==0){
            h[i]=0;
        }else{
            if(i==0 || h[i-1]<=1){
                int t1=i,t2=SA[rank[i]-1],cc=0;
                for(int j=0;t1+j<n && t2+j<n;j++){
                    if(T[t1+j]==T[t2+j]){
                        cc++;
                    }else{
                        break;
                    }
                }
                h[i]=cc;
            }else{
                int t1=i,t2=SA[rank[i]-1],cc=h[i-1]-1;
                for(int j=h[i-1]-1;t1+j<n && t2+j<n;j++){
                    if(T[t1+j]==T[t2+j]){
                        cc++;
                    }else{
                        break;
                    }
                }
                h[i]=cc;
```

```
            }
        }
    }
    for(int i=0;i<n;i++) height[i]=h[SA[i]];
    delete [] rank;delete [] h;
}
```

其它

分数

```
#include <cmath>

struct Frac {
    int num, den;
};

int gcd(int a, int b) {
    int t;
    if (a < 0) {
        a = -a;
    }
    if (b < 0) {
        b = -b;
    }
    if (!b) {
        return a;
    }
    while (t = a % b) {
        a = b;
        b = t;
    }
    return b;
}

void simplify(Frac & f) {
    int t;
    if (t = gcd(f.num, f.den)) {
        f.num /= t;
        f.den /= t;
    } else {
        f.den = 1;
    }
}

Frac f(int n, int d, int s = 1) {
    Frac ret;
    if (d < 0) {
        ret.num = -n;
        ret.den = -d;
    } else {
        ret.num = n;
        ret.den = d;
```

```
    }
    if (s) {
        simplify(ret);
    }
    return ret;
}

Frac convert(double x) {
    Frac ret;
    for (ret.den = 1; fabs(x - int(x)) > 1e-10; ret.den *= 10, x *= 10);
    ret.num = (int)x;
    simplify(ret);
    return ret;
}

int fraqcmp(Frac a, Frac b) {
    int g1 = gcd(a.den, b.den), g2 = gcd(a.num, b.num);
    if (!g1 || !g2) {
        return 0;
    }
    return b.den / g1 *(a.num / g2) - a.den / g1 *(b.num / g2);
}

Frac add(Frac a, Frac b) {
    int g1 = gcd(a.den, b.den), g2, t;
    if (!g1) {
        return f(1, 0, 0);
    }
    t = b.den / g1 * a.num + a.den / g1 * b.num;
    g2 = gcd(g1, t);
    return f(t / g2, a.den / g1 *(b.den / g2), 0);
}

Frac sub(Frac a, Frac b) {
    return add(a, f(-b.num, b.den, 0));
}

Frac mul(Frac a, Frac b) {
    int t1 = gcd(a.den, b.num), t2 = gcd(a.num, b.den);
    if (!t1 || !t2) {
        return f(1, 1, 0);
    }
    return f(a.num / t2 *(b.num / t1), a.den / t1 *(b.den / t2), 0);
}

Frac div(Frac a, Frac b) {
    return mul(a, f(b.den, b.num, 0));
```

```
}
```

矩阵

```
#include <cmath>
const int MAXN = 100;

#define zero(x) (fabs(x) < 1e-10)

struct mat {
    int n, m;
    double data[MAXN][MAXN];
};

bool mul(mat & c, const mat & a, const mat & b) {
    int i, j, k;
    if (a.m != b.n) {
        return false;
    }
    c.n = a.n;
    c.m = b.m;
    for (i = 0; i < c.n; i++) {
        for (j = 0; j < c.m; j++) {
            for (c.data[i][j] = k = 0; k < a.m; k++) {
                c.data[i][j] += a.data[i][k] * b.data[k][j];
            }
        }
    }
    return true;
}

bool inv(mat & a) {
    int i, j, k, is[MAXN], js[MAXN];
    double t;
    if (a.n != a.m) {
        return false;
    }
    for (k = 0; k < a.n; k++) {
        for (t = 0, i = k; i < a.n; i++) {
            for (j = k; j < a.n; j++) {
                if (fabs(a.data[i][j]) > t) {
                    t = fabs(a.data[is[k] = i][js[k] = j]);
                }
            }
        }
        if (zero(t)) {
            return false;
        }
```

```
        if (is[k] != k) {
            for (j = 0; j < a.n; j++) {
                t = a.data[k][j];
                a.data[k][j] = a.data[is[k]][j];
                a.data[is[k]][j] = t;
            }
        }
        if (js[k] != k) {
            for (i = 0; i < a.n; i++) {
                t = a.data[i][k];
                a.data[i][k] = a.data[i][js[k]];
                a.data[i][js[k]] = t;
            }
        }
        a.data[k][k] = 1 / a.data[k][k];
        for (j = 0; j < a.n; j++) {
            if (j != k) {
                a.data[k][j] *= a.data[k][k];
            }
        }
        for (i = 0; i < a.n; i++) {
            if (i != k) {
                for (j = 0; j < a.n; j++) {
                    if (j != k) {
                        a.data[i][j] -= a.data[i][k] * a.data[k][j];
                    }
                }
            }
        }
        for (i = 0; i < a.n; i++) {
            if (i != k) {
                a.data[i][k] *= -a.data[k][k];
            }
        }
    }
    for (k = a.n - 1; k >= 0; k--) {
        for (j = 0; j < a.n; j++) {
            if (js[k] != k) {
                t = a.data[k][j];
                a.data[k][j] = a.data[js[k]][j];
                a.data[js[k]][j] = t;
            }
        }
        for (i = 0; i < a.n; i++) {
            if (is[k] != k) {
                t = a.data[i][k];
                a.data[i][k] = a.data[i][is[k]];
```

```
                a.data[i][is[k]] = t;
            }
        }
    }
    return true;
}

double det(const mat & a) {
    int i, j, k, sign = 0;
    double b[MAXN][MAXN], ret = 1, t;
    if (a.n != a.m) {
        return 0;
    }
    for (i = 0; i < a.n; i++) {
        for (j = 0; j < a.m; j++) {
            b[i][j] = a.data[i][j];
        }
    }
    for (i = 0; i < a.n; i++) {
        if (zero(b[i][i])) {
            for (j = i + 1; j < a.n; j++) {
                if (!zero(b[j][i])) {
                    break;
                }
            }
            if (j == a.n) {
                return 0;
            }
            for (k = i; k < a.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 < a.n; k++) {
            b[i][k] /= b[i][i];
        }
        for (j = i + 1; j < a.n; j++) {
            for (k = i + 1; k < a.n; k++) {
                b[j][k] -= b[j][i] * b[i][k];
            }
        }
    }
    if (sign & 1) {
        ret = -ret;
    }
    return ret;
```

```
}
```

整系数不定方程

```
/*
  说明：解整系数不定方程模块
  使用方法：
    AX=B 输入矩阵 a,b
        a 是 m 行 n 列矩阵,b 为 m 维列向量
        a,b 元素均为整数
        调用 cal(a,b,m,n,p)
        返回一个 int 表示解中的自由未知数的个数。
        如果返回值为-1 则原方程组无整数解。
        否则设返回值为 r
        参数 p 用来返回一个 n 行(r+1)列的方阵
    原方程的整数通解形式是
        P*V
        V 是一个(r+1)维列向量:V=[t0,t1,t2...tr-1,1]
        其中 t0,t1..tr-1 为 r 个任意的整数
        也可利用此求出非 0 整数解:
        (1) 如果 P 的最后一列非 0 则可把取 t0=t1=t2=...=tr-1=0 即可
        (2) 否则 如果 p(i,j)非 0 则取 tj=1 其余 ti (i!=j)都取 0 即可
        (3) 否则只有 0 解
*/

/*
    注意:
          (1) 数组 a,b 都被改变
          (2) 数组 a 需要至少可以容纳 (m+n)*(n+1)个元素
          (3) 数组 p 需要至少可以容纳 (n+1)*(n+1)个元素
          (4) 如果 n 很大(未知数个数太多),则数组的元素可能超出 int 必要时 使用 long long 或 bignum
*/

#define type int
#define N 205
void changecol(type a[][N],int c1,int c2,int m) {
int i;
type t;
    for (i=0;i<m;++i) {
        t=a[i][c1];
        a[i][c1]=a[i][c2];
        a[i][c2]=t;
    }
}

void changerow(type a[][N],int r1,int r2,int n) {
int i;
type t;
```

```
    for (i=0;i<n;++i) {
        t=a[r1][i];
        a[r1][i]=a[r2][i];
        a[r2][i]=t;
    }
}

void div(type a[][N],int c1,int c2,int i,int m) { // c1 c2 列辗转相除
int j,t;
type k;

    for (;(a[i][c1]!=0) && (a[i][c2]!=0);t=c1,c1=c2,c2=t) {
        k=a[i][c1]/a[i][c2];
        for (j=i;j<m;++j) {
            a[j][c1]-=a[j][c2]*k;
        }
    }
}

int cal(type a[][N],type *b,int m,int n,type p[][N]) {
int r,i,j,k,h,mm,nn;
    for (i=0;i<n;++i) {
        for (j=0;j<n;++j) {
            a[m+i][j]=0;
        }
        a[m+i][i]=1;
    }
    for (i=0;i<m;++i) {
        a[i][n]=b[i];
    }
    for (i=0;i<n;++i) {
        a[m+i][n]=0;
    }
    mm=m+n;
    nn=n+1;
    r=m;
    for (i=0,j=n-1;(i<r) && (j>=0);++i,--j) { //要把第(i,j)变成 1
        for (h=0;h<=j;++h) {
            if (a[i][h]!=0) {
                break;
            }
        }
        for (k=h+1;k<=j;++k) {
            if (a[i][k]==0) {
                changecol(a,h++,k,mm);
            }
```

```
        }
        if (h>j) { //全是0
            if (a[i][n]!=0) {
                return -1;
            }
            changerow(a,i--,--r,nn);
            ++j;
            continue;
        }
        for (;h<j;++h) {
            div(a,h,h+1,i,mm);
            if (a[i][h]!=0) {
                changecol(a,h,h+1,mm);
            }
        }
        if (a[i][n]%a[i][j]!=0) {
            return -1;
        }
        a[i][n]/=a[i][j];
        a[i][j]=1;
        for (k=i+1;k<r;++k) {
            a[k][n]-=a[i][n]*a[k][j];
            a[k][j]=0;
        }
    }
    for (j=i;j<r;++j) {
        if (a[j][n]!=0) {
            return -1;
        }
    }
    r=i;
    k=n-r;
    for (i=0;i<n;++i) {
        for (j=0;j<k;++j) {
            p[i][j]=a[m+i][j];
        }
    }
    for (i=0;i<n;++i) {
        p[i][k]=0;
        for (j=0;j<r;++j) {
            p[i][k]+=a[j][n]*p[i][n-1-j];
        }
    }
    return k;
}
```

带求模行列式

```cpp
// 传入矩阵mat,大小n,被模数m,返回|mat| % m
#include <cstring>
#include <algorithm>
using namespace std;

int extGcd(int a, int b, int & x, int & y)
{
    int t, ret;
    if (!b) {
        x = 1;
        y = 0;
        return a;
    }
    ret = extGcd(b, a % b, x, y);
    t = x;
    x = y;
    y = t - a / b * y;
    return ret;
}

int modInverse(int a, int n, int notused)
{
    int x, y;
    extGcd(n, a, x, y);
    return y;
}

int mod(int a, int n)
{
    return (a % n + n) % n;
}

int gcd(int a, int b)
{
    return a % b ? gcd(b, a % b) : b;
}

int mulMod(int a, int b, int n)
{
    return mod((long long) a * b % n, n);
}

int powMod(int a, int b, int n)
{
    int ret = 1;
    for (; b; b >>= 1, a = (int) ((long long) a * a % n)) {
        if (b & 1) {
```

```
            ret = (int) ((long long) ret * a % n);
        }
    }
    return ret;
}

/*
    @author crazyboy
    a/b(mod p) 是可以转化成 a*inv(b)(mod p) 的， inv(b)*b==1(mod p) ，然后把 M 分成
P1^Q1 * P2 ^ Q2,
    分别算出%(P1^Q1),%(P2^Q2)...，最后用中国剩余定理合并。
    如果是%P(P is a prime)，那么很简单，直接把/b 弄成*inv(b)，就可以了，否则会很麻烦。
    这一点才是这道题目主要的难点，处理方法就是选主元时如果某一列是一个 P 的倍数，就都除去，然后
%P^(Q-1)。
*/

const int MAXN = 100;

int detMod(int mat[][MAXN], int n, int m)
{
    if(m == 1)
        return 0;
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++j)
            mat[i][j] = mod(mat[i][j], m);
    int f[30], c[30], fs = 0;
    int tmp = m;
    for(int i = 2; i * i <= tmp; ++i)
        if(tmp % i == 0) {
            c[fs] = 0;
            f[fs] = i;
            while(tmp % i == 0) {
                tmp /= i;
                ++c[fs];
            }
            ++fs;
        }
    if(tmp != 1) {
        c[fs] = 1;
        f[fs++] = tmp;
    }
    if(fs != 1){
        int a[30], b[30];
        int nmat[MAXN][MAXN];
        for(int i = 0; i < fs; ++i) {
            a[i] = powMod(f[i], c[i], m);
            for(int j = 0; j < n; ++j)
```

```
                memcpy(nmat[j], mat[j], sizeof(mat[0]));
            b[i] = detMod(nmat, n, a[i]);
        }
        int ret = 0;
        for(int i = 0; i < fs; ++i){
            int e = modInverse(m / a[i], a[i], a[i] - a[i] / f[i]) * (m / a[i]);
            ret = mod(ret + mulMod(e, b[i], m), m);
        }
        return ret;
    }
    int em = m - m / f[0];
    int ret = 1;
    for(int i = 0; i < n; ++i) {
        int r = i, gr = m + 1;
        while(r < n && mat[r][i] == 0)
            ++r;
        if(r == n)
            return 0;
        r = -1;
        for(int j = i; j < n; ++j) {
            int g = gcd(mat[j][i], m);
            if(g != 0 && g < gr) {
                gr = g;
                r = j;
            }
        }
        if(gr != 1) {
            for(int j = 0; j < n; ++j)
                mat[j][i] /= gr;
            return mulMod(ret, gr * detMod(mat, n, m / gr), m);
        }
        if(r != i){
            ret = mod(-ret, m);
            for(int j = 0; j < n; ++j)
                swap(mat[i][j], mat[r][j]);
        }
        ret = mulMod(ret, mat[i][i], m);
        int inv = modInverse(mat[i][i], m, em);
        for(int j = i; j < n; ++j)
            mat[i][j] = mulMod(mat[i][j], inv, m);
        for(int j = 0; j < n; ++j)
            if(j != i && mat[j][i]) {
                for(int k = n - 1; k >= i; --k)
                    mat[j][k] = mod(mat[j][k] - mulMod(mat[j][i], mat[i][k], m), m)
;
            }
    }
```

```
    return ret;
}
```

线性方程组

```
#include <cmath>

const int MAXN = 100;
const double EPS = 1e-10;

//列主元gauss消去求解a[][]x[]=b[]
//返回是否有唯一解,若有解在b[]中
bool gaussCpivot(int n, double a[][MAXN], double b[]) {
    int i, j, k, row;
    double maxp, t;
    for (k = 0; k < n; k++) {
        for (maxp = 0, i = k; i < n; i++) {
            if (fabs(a[i][k]) > fabs(maxp)) {
                maxp = a[row = i][k];
            }
        }
        if (fabs(maxp) < EPS) {
            return false;
        }
        if (row != k) {
            for (j = k; j < n; j++) {
                t = a[k][j];
                a[k][j] = a[row][j];
                a[row][j] = t;
            }
            t = b[k];
            b[k] = b[row];
            b[row] = t;
        }
        for (j = k + 1; j < n; j++) {
            a[k][j] /= maxp;
            for (i = k + 1; i < n; i++) {
                a[i][j] -= a[i][k] * a[k][j];
            }
        }
        b[k] /= maxp;
        for (i = k + 1; i < n; i++) {
            b[i] -= b[k] * a[i][k];
        }
    }
    for (i = n - 1; i >= 0; i--) {
        for (j = i + 1; j < n; j++) {
            b[i] -= a[i][j] * b[j];
```

```
        }
    }
    return true;
}

//全主元 gauss 消去解 a[][]x[]=b[]
//返回是否有唯一解,若有解在 b[]中
bool gaussTpivot(int n, double a[][MAXN], double b[]) {
    int i, j, k, row, col, index[MAXN];
    double maxp, t;
    for (i = 0; i < n; i++) {
        index[i] = i;
    }
    for (k = 0; k < n; k++) {
        for (maxp = 0, i = k; i < n; i++) {
            for (j = k; j < n; j++) {
                if (fabs(a[i][j]) > fabs(maxp)) {
                    maxp = a[row = i][col = j];
                }
            }
        }
        if (fabs(maxp) < EPS) {
            return false;
        }
        if (col != k) {
            for (i = 0; i < n; i++) {
                t = a[i][col];
                a[i][col] = a[i][k];
                a[i][k] = t;
            }
            j = index[col];
            index[col] = index[k];
            index[k] = j;
        }
        if (row != k) {
            for (j = k; j < n; j++) {
                t = a[k][j];
                a[k][j] = a[row][j];
                a[row][j] = t;
            }
            t = b[k];
            b[k] = b[row];
            b[row] = t;
        }
        for (j = k + 1; j < n; j++) {
            a[k][j] /= maxp;
            for (i = k + 1; i < n; i++) {
```

```
                a[i][j] -= a[i][k] * a[k][j];
            }
        }
        b[k] /= maxp;
        for (i = k + 1; i < n; i++) {
            b[i] -= b[k] * a[i][k];
        }
    }
    for (i = n - 1; i >= 0; i--) {
        for (j = i + 1; j < n; j++) {
            b[i] -= a[i][j] * b[j];
        }
    }
    for (k = 0; k < n; k++) {
        a[0][index[k]] = b[k];
    }
    for (k = 0; k < n; k++) {
        b[k] = a[0][k];
    }
    return true;
}
```

线性相关

```
//判线性相关(正交化)
//传入 m 个 n 维向量
#include <cmath>
const int MAXN = 100;
const double EPS = 1e-10;

bool linearDependent(int m, int n, double vec[][MAXN]) {
    double ort[MAXN][MAXN], e;
    int i, j, k;
    if (m > n) {
        return true;
    }
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            ort[i][j] = vec[i][j];
        }
        for (k = 0; k < i; k++) {
            for (e = j = 0; j < n; j++) {
                e += ort[i][j] * ort[k][j];
            }
            for (j = 0; j < n; j++) {
                ort[i][j] -= e * ort[k][j];
            }
            for (e = j = 0; j < n; j++) {
```

```
                e += ort[i][j] * ort[i][j];
            }
            if (fabs(e = sqrt(e)) < EPS) {
                return 1;
            }
            for (j = 0; j < n; j++) {
                ort[i][j] /= e;
            }
        }
    }
    return false;
}
```

日期

```
int days[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

class Date {
public:
    //判闰年
    inline static bool isLeap(int year) {
        return (year % 4 == 0 && year % 100 != 0) || year % 400 == 0;
    }

    int year, month, day;

    //判合法性
    inline bool isLegal() const {
        if (month <= 0 || month > 12) {
            return false;
        } if (month == 2) {
            return day > 0 && day <= 28 + isLeap(year);
        }
        return day > 0 && day <= days[month - 1];
    }

    //比较日期大小
    inline int compareTo(const Date & other) const {
        if (year != other.year) {
            return year - other.year;
        }
        if (month != other.month) {
            return month - other.month;
        }
        return day - other.day;
    }

    //返回指定日期是星期几  0 (Sunday) ... 6 (Saturday)
```

```
    inline int toWeekday() const {
        int tm = month >= 3 ? (month - 2) : (month + 10);
        int ty = month >= 3 ? year : (year - 1);
        return (ty + ty / 4 - ty / 100 + ty / 400 + (int)(2.6 * tm - 0.2) + day) % 7;
    }

    //日期转天数偏移
    inline int toInt() const {
        int ret = year * 365 + (year - 1) / 4 - (year - 1) / 100 + (year - 1) / 400;
        days[1] += isLeap(year);
        for (int i = 0; i < month - 1; ret += days[i++]);
        days[1] = 28;
        return ret + day;
    }

    //天数偏移转日期
    inline void fromInt(int a) {
        year = a / 146097 * 400;
        for (a %= 146097; a >= 365 + isLeap(year); a -= 365 + isLeap(year), year++);
        days[1] += isLeap(year);
        for (month = 1; a >= days[month - 1]; a -= days[month - 1], month++);
        days[1] = 28;
        day = a + 1;
    }
};
```

位操作

功能	示例	位运算	备注
把右数第 k 位置 0		x & (~(1 << k))	k 从 0 开始计数
把右数第 k 位置 1		x | (1 << k)	k 从 0 开始计数
把右数第 k 位取反		x ^ (1 << k)	k 从 0 开始计数
得到右数第 k 位值		(x >> k) & 1	k 从 0 开始计数
得到末尾 k 位		x & ((1 << k) - 1)	
把最右边的 1 置 0	01011000 -> 01010000	x & (x - 1)	把结果跟 0 作比较可以得出 x 是否为 2 的幂
得到最右边的 1 的掩码	01011000 -> 00001000	x & (-x)	
把最右边的 0 置 1	01011000 -> 01011001	x | (x + 1)	
得到最右边的 0 的掩码	10100111 -> 00001000	(~x) & (x + 1)	
把末尾连续 0 串置 1	01011000 -> 01011111	x | (x - 1)	如果 x 为 0 则结果为全 1
得到末尾连续 0 串的掩码	01011000 -> 00000111	(~x) & (x - 1)	或者使用 ~(x | (-x)) 和 (x & (-x)) - 1

得到最右边的 1 及其右边连续 0 串的掩码	01011000 -> 00001111	x ^ (x - 1)	如果 x 为 0 则结果为全 1
把最右边的连续 1 串置 0	01011000 -> 01000000	((x | (x - 1)) + 1) & x	把结果跟 0 作比较可以得出 x 是否为 $2^j - 2^k$ (j ≥ k ≥ 0)
得到最右边的连续 1 串的掩码	01011000 -> 00011000	(((x | (x - 1)) + 1) & x) ^ x	
下舍入到 2^k 倍数		x & ((-1) << k)	
上舍入到 2^k 倍数		t = (1 << k) - 1 x = (x + t) & (~t)	

```
// 遍历一个掩码的所有子集掩码,不包括 0 和其自身
// 传入表示超集的掩码
void iterateSubset(int mask)
{
    for(int sub = (mask - 1) & mask; sub > 0; sub = (sub - 1) & mask)
    {
        int incsub = ~sub & mask; // 递增顺序的子集
        // gogogo
    }
}

// 下一个包含同样数量二进制 1 的掩码
// 传入掩码,掩码不能为 0
unsigned snoob(unsigned mask)
{
    unsigned smallest, ripple, ones;
                                    // x = xxx0 1111 0000
    smallest = mask & -mask;        //      0000 0001 0000
    ripple = mask + smallest;       //      xxx1 0000 0000
    ones = mask ^ ripple;           //      0001 1111 0000
    ones = (ones >> 2) / smallest;  //      0000 0000 0111
    return ripple | ones;           //      xxx1 0000 0111
}

// 遍历{0, 1, … , n-1}的所有 k 元子集
// 传入 n 和 k,k 不为 0
void iterateSubset(int n, int k)
{
    int s = (1 << k) - 1;
    while (!(s & 1 << n)) {
        // gogogo
        s = snoob(s);
    }
}

// 求一个 32 位整数二进制 1 的位数
```

```
// 初始化函数先调用一次

int ones[256];

void initOnes()
{
    for (int i = 1; i < 256; ++i)
        ones[i] = ones[i & (i - 1)] + 1;
}

int countOnes(int n)
{
    return ones[n & 255] + ones[(n >> 8) & 255] + ones[(n >> 16) & 255] + ones[(n >
> 24) & 255];
}

// 求一个 32 位整数二进制 1 的位数的奇偶性
// 偶数返回 0,奇数返回 1
int parityOnes(unsigned n)
{
    n ^= n >> 1;
    n ^= n >> 2;
    n ^= n >> 4;
    n ^= n >> 8;
    n ^= n >> 16;
    return n & 1; // n 的第 i 位是原数第 i 位到最左侧位的奇偶性
}

// 对一个 32 位整数进行位反转
unsigned revBits(unsigned n)
{
    n = (n & 0x55555555) << 1 | (n >> 1) & 0x55555555;
    n = (n & 0x33333333) << 2 | (n >> 2) & 0x33333333;
    n = (n & 0x0F0F0F0F) << 4 | (n >> 4) & 0x0F0F0F0F;
    n = (n << 24) | ((n & 0xFF00) << 8) | ((n >> 8) & 0xFF00) | (n >> 24);
    return n;
}

// 对 n 个数根据二进制掩码求部分和，类似的可以用于求 gcd，求二进制 1 的个数等等

void partSum(int n, int a[], int s[])
{
    s[0] = 0;
    for (int i = 0; i < n; i++) {
        s[1 << i] = a[i];
    }
    for (int i = 1; i < (1 << n); i++) {
```

```
        s[i] = s[i&(i-1)] + s[i^(i&(i-1))];
    }
}
```

附录_结构

FixedList

```
template <class T,int maxSize>
class FixedList{
private:
    T arr[maxSize];
    int begin,end;
public:
    FixedList(){
        clear();
    }

    inline void clear(){
        begin=0;
        end=0;
    }

    inline void push_front(const T & elem){
        begin--;
        if(begin<0){
            begin=maxSize-1;
        }
        arr[begin]=elem;
    }

    inline void push_back(const T & elem){
        arr[end++]=elem;
        if(end==maxSize){
            end=0;
        }
    }

    inline void pop_front(){
        begin++;
        if(begin==maxSize){
            begin=0;
        }
    }

    inline void pop_back(){
        end--;
        if(end<0){
            end=maxSize-1;
        }
```

```
    }

    inline const T & front() const{
        return arr[begin];
    }

    inline const T & back() const{
        return arr[(end==0?maxSize-1:end-1)];
    }

    //check the correctness the index beforehand
    inline const T & operator [] (int index) const{
        int tmpindex=index+begin;
        if(tmpindex>=maxSize) tmpindex-=maxSize;
        return arr[tmpindex];
    }

    inline bool empty() const{
        return begin==end;
    }

    inline int size() const{
        return (begin<=end?end-begin:end+maxSize-begin);
    }
};

FixedList<int,maxl> snake[maxm];
```

FixedQueue

```
// Should be global if maxSize is large
// maxSize should be more than the largest number of concurrent elements
template <class T, int maxSize>
class FixedQueue {
private:
   T arr[maxSize];
   int begin, end;

public:
   FixedQueue() {
      clear();
   }

   inline void clear() {
      begin = 0;
      end = 0;
   }
```

```
    inline void push(const T & elem) {
        arr[end++] = elem;
        if (end == maxSize) {
            end = 0;
        }
    }

    inline const T & front() const {
        return arr[begin];
    }

    inline void pop() {
        if (++begin == maxSize) {
            begin = 0;
        }
    }

    inline bool empty() const {
        return begin == end;
    }

    inline const T & back() const {
        return arr[(end == 0 ? maxSize - 1 : end - 1)];
    }

    inline int size() const {
        return (begin <= end ? end - begin : end + maxSize - begin);
    }
};

FixedQueue<int, MAXN * MAXN * MAXN> bfs;
```

线段树

线段树应用

- 求面积
  - 坐标离散化
  - 垂直边按 x 坐标排序
  - 从左往右用线段树处理垂直边

    累计每个离散 x 区间长度和线段树长度的乘积
- 求周长
  - 坐标离散化
  - 垂直边按 x 坐标排序，第二关键字为入边优于出边
  - 从左往右用线段树处理垂直边

  在每个离散点上先加入所有入边，累计线段树长度变化值

  再删除所有出边，累计线段树长度变化值
  - 水平边按 y 坐标排序，第二关键字为入边优于出边

♦ 从上往下用线段树处理水平边

在每个离散点上先加入所有入边，累计线段树长度变化值

再删除所有出边，累计线段树长度变化值

```
//线段树
//可以处理加入边和删除边不同的情况
//incSet 和 decSeg 用于加入边
//segLen 求长度
//t 传根节点(一律为 1)
//L0, r0 传树的节点范围(一律为 1..t)
//L, r 传线段(端点)
const int MAXN = 10000;
class SegTree {
    int n, cnt[MAXN], len[MAXN];

    SegTree(int t) : n(t) {
        for (int i = 1; i <= t; i++) {
            cnt[i] = len[i] = 0;
        }
    };

    void update(int t, int L, int r);
    void incSet(int t, int L0, int r0, int L, int r);
    void decSeg(int t, int L0, int r0, int L, int r);
    int segLen(int t, int L0, int r0, int L, int r);
};

inline int length(int L, int r) {
    return r - L;
}

void SegTree::update(int t, int L, int r) {
    if (cnt[t] || r - L == 1) {
        len[t] = length(L, r);
    } else {
        len[t] = len[t + t] + len[t + t + 1];
    }
}

void SegTree::incSet(int t, int L0, int r0, int L, int r) {
    if (L0 == L && r0 == r) {
        cnt[t]++;
    }
    else{
        int m0 = (L0 + r0) >> 1;
        if (L < m0) {
            incSet(t + t, L0, m0, L, m0 < r ? m0 : r);
```

```
        }
        if (r > m0) {
            incSet(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
        }
        if (cnt[t + t] && cnt[t + t + 1]) {
            cnt[t + t]--;
            update(t + t, L0, m0);
            cnt[t + t + 1]--;
            update(t + t + 1, m0, r0);
            cnt[t]++;
        }
    }
    update(t, L0, r0);
}

void SegTree::decSeg(int t, int L0, int r0, int L, int r) {
    if (L0 == L && r0 == r) {
        cnt[t]--;
    } else if (cnt[t]) {
        cnt[t]--;
        if (L > L0) {
            incSet(t, L0, r0, L0, L);
        }
        if (r < r0) {
            incSet(t, L0, r0, r, r0);
        }
    } else {
        int m0 = (L0 + r0) >> 1;
        if (L < m0) {
            decSeg(t + t, L0, m0, L, m0 < r ? m0 : r);
        }
        if (r > m0) {
            decSeg(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
        }
    }
    update(t, L0, r0);
}

int SegTree::segLen(int t, int L0, int r0, int L, int r) {
    if (cnt[t] || (L0 == L && r0 == r)) {
        return len[t];
    } else {
        int m0 = (L0 + r0) >> 1, ret = 0;
        if (L < m0) {
            ret += segLen(t + t, L0, m0, L, m0 < r ? m0 : r);
        }
        if (r > m0) {
```

```
            ret += segLen(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
        }
        return ret;
    }
}
```

线段树扩展

```
//线段树扩展
//可以计算长度和线段数
//可以处理加入边和删除边不同的情况
//incSeg 和 decSeg 用于加入边
//segLen 求长度, setCut 求线段数
//t 传根节点(一律为 1)
//L0, r0 传树的节点范围(一律为 1..t)
//L, r 传线段(端点)
const int MAXN = 10000;
class SegTree {
public:
    int n, cnt[MAXN], len[MAXN], cut[MAXN], bl[MAXN], br[MAXN];

    SegTree(int t) : n(t) {
        for (int i = 1; i <= t; i++) {
            cnt[i] = len[i] = cut[i] = bl[i] = br[i] = 0;
        }
    };

    void update(int t, int L, int r);
    void incSeg(int t, int L0, int r0, int L, int r);
    void decSeg(int t, int L0, int r0, int L, int r);
    int segLen(int t, int L0, int r0, int L, int r);
    int setCut(int t, int L0, int r0, int L, int r);
};

inline int length(int L, int r) {
    return r - L;
}

void SegTree::update(int t, int L, int r) {
    if (cnt[t] || r - L == 1) {
        len[t] = length(L, r);
        cut[t] = bl[t] = br[t] = 1;
    } else {
        len[t] = len[t + t] + len[t + t + 1];
        cut[t] = cut[t + t] + cut[t + t + 1];
        if (br[t + t] && bl[t + t + 1]) {
            cut[t]--;
        }
```

```
        bl[t] = bl[t + t];
        br[t] = br[t + t + 1];
    }
}

void SegTree::incSeg(int t, int L0, int r0, int L, int r) {
    if (L0 == L && r0 == r) {
        cnt[t]++;
    } else {
        int m0 = (L0 + r0) >> 1;
        if (L < m0) {
            incSeg(t + t, L0, m0, L, m0 < r ? m0 : r);
        }
        if (r > m0) {
            incSeg(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
        }
        if (cnt[t + t] && cnt[t + t + 1]) {
            cnt[t + t]--;
            update(t + t, L0, m0);
            cnt[t + t + 1]--;
            update(t + t + 1, m0, r0);
            cnt[t]++;
        }
    }
    update(t, L0, r0);
}

void SegTree::decSeg(int t, int L0, int r0, int L, int r) {
    if (L0 == L && r0 == r) {
        cnt[t]--;
    } else if (cnt[t]) {
        cnt[t]--;
        if (L > L0) {
            incSeg(t, L0, r0, L0, L);
        }
        if (r < r0) {
            incSeg(t, L0, r0, r, r0);
        }
    } else {
        int m0 = (L0 + r0) >> 1;
        if (L < m0) {
            decSeg(t + t, L0, m0, L, m0 < r ? m0 : r);
        }
        if (r > m0) {
            decSeg(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
        }
    }
```

```
    update(t, L0, r0);
}

int SegTree::segLen(int t, int L0, int r0, int L, int r) {
    if (cnt[t] || (L0 == L && r0 == r)) {
        return len[t];
    }
    else{
        int m0 = (L0 + r0) >> 1, ret=0;
        if (L < m0) {
            ret += segLen(t + t, L0, m0, L, m0 < r ? m0 : r);
        }
        if (r > m0) {
            ret += segLen(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
        }
        return ret;
    }
}

int SegTree::setCut(int t, int L0, int r0, int L, int r) {
    if (cnt[t]) {
        return 1;
    }
    if (L0 == L && r0 == r) {
        return cut[t];
    } else {
        int m0 = (L0 + r0) >> 1, ret = 0;
        if (L < m0) {
            ret += setCut(t + t, L0, m0, L, m0 < r ? m0 : r);
        }
        if (r > m0) {
            ret += setCut(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
        }
        if (L < m0 && r > m0 && br[t + t] && bl[t + t + 1]) {
            ret--;
        }
        return ret;
    }
}
```

附录_应用

大数(整数类封装)

```
// 不推荐使用，最好用 Java

#include <iostream.h>
#include <string.h>

#define DIGIT    4
#define DEPTH    10000
#define MAX      100
typedef int bignum_t[MAX+1];

int read(bignum_t a,istream& is=cin){
    char buf[MAX*DIGIT+1],ch;
    int i,j;
    memset((void*)a,0,sizeof(bignum_t));
    if (!(is>>buf))     return 0;
    for (a[0]=strlen(buf),i=a[0]/2-1;i>=0;i--)
        ch=buf[i],buf[i]=buf[a[0]-1-i],buf[a[0]-1-i]=ch;
    for (a[0]=(a[0]+DIGIT-1)/DIGIT,j=strlen(buf);j<a[0]*DIGIT;buf[j++]='0');
    for (i=1;i<=a[0];i++)
        for (a[i]=0,j=0;j<DIGIT;j++)
            a[i]=a[i]*10+buf[i*DIGIT-1-j]-'0';
    for (;!a[a[0]]&&a[0]>1;a[0]--);
    return 1;
}

void write(const bignum_t a,ostream& os=cout){
    int i,j;
    for (os<<a[i=a[0]],i--;i;i--)
        for (j=DEPTH/10;j;j/=10)
            os<<a[i]/j%10;
}

int comp(const bignum_t a,const bignum_t b){
    int i;
    if (a[0]!=b[0])
        return a[0]-b[0];
    for (i=a[0];i;i--)
        if (a[i]!=b[i])
            return a[i]-b[i];
    return 0;
}
```

```
int comp(const bignum_t a,const int b){
    int c[12]={1};
    for (c[1]=b;c[c[0]]>=DEPTH;c[c[0]+1]=c[c[0]]/DEPTH,c[c[0]]%=DEPTH,c[0]++);
    return comp(a,c);
}

int comp(const bignum_t a,const int c,const int d,const bignum_t b){
    int i,t=0,O=-DEPTH*2;
    if (b[0]-a[0]<d&&c)
        return 1;
    for (i=b[0];i>d;i--){
        t=t*DEPTH+a[i-d]*c-b[i];
        if (t>0) return 1;
        if (t<O) return 0;
    }
    for (i=d;i;i--){
        t=t*DEPTH-b[i];
        if (t>0) return 1;
        if (t<O) return 0;
    }
    return t>0;
}

void add(bignum_t a,const bignum_t b){
    int i;
    for (i=1;i<=b[0];i++)
        if ((a[i]+=b[i])>=DEPTH)
            a[i]-=DEPTH,a[i+1]++;
    if (b[0]>=a[0])
        a[0]=b[0];
    else
        for (;a[i]>=DEPTH&&i<a[0];a[i]-=DEPTH,i++,a[i]++);
    a[0]+=(a[a[0]+1]>0);
}

void add(bignum_t a,const int b){
    int i=1;
    for (a[1]+=b;a[i]>=DEPTH&&i<a[0];a[i+1]+=a[i]/DEPTH,a[i]%=DEPTH,i++);
    for (;a[a[0]]>=DEPTH;a[a[0]+1]=a[a[0]]/DEPTH,a[a[0]]%=DEPTH,a[0]++);
}

void sub(bignum_t a,const bignum_t b){
    int i;
    for (i=1;i<=b[0];i++)
        if ((a[i]-=b[i])<0)
            a[i+1]--,a[i]+=DEPTH;
    for (;a[i]<0;a[i]+=DEPTH,i++,a[i]--);
```

```
    for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void sub(bignum_t a,const int b){
    int i=1;
    for (a[1]-=b;a[i]<0;a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEP
TH,i++);
    for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void sub(bignum_t a,const bignum_t b,const int c,const int d){
    int i,O=b[0]+d;
    for (i=1+d;i<=O;i++)
        if ((a[i]-=b[i-d]*c)<0)
            a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH;
    for (;a[i]<0;a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH,i++)
;
    for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void mul(bignum_t c,const bignum_t a,const bignum_t b){
    int i,j;
    memset((void*)c,0,sizeof(bignum_t));
    for (c[0]=a[0]+b[0]-1,i=1;i<=a[0];i++)
        for (j=1;j<=b[0];j++)
            if ((c[i+j-1]+=a[i]*b[j])>=DEPTH)
                c[i+j]+=c[i+j-1]/DEPTH,c[i+j-1]%=DEPTH;
    for (c[0]+=(c[c[0]+1]>0);!c[c[0]]&&c[0]>1;c[0]--);
}

void mul(bignum_t a,const int b){
    int i;
    for (a[1]*=b,i=2;i<=a[0];i++){
        a[i]*=b;
        if (a[i-1]>=DEPTH)
            a[i]+=a[i-1]/DEPTH,a[i-1]%=DEPTH;
    }
    for (;a[a[0]]>=DEPTH;a[a[0]+1]=a[a[0]]/DEPTH,a[a[0]]%=DEPTH,a[0]++);
    for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void mul(bignum_t b,const bignum_t a,const int c,const int d){
    int i;
    memset((void*)b,0,sizeof(bignum_t));
    for (b[0]=a[0]+d,i=d+1;i<=b[0];i++)
        if ((b[i]+=a[i-d]*c)>=DEPTH)
            b[i+1]+=b[i]/DEPTH,b[i]%=DEPTH;
```

```
    for (;b[b[0]+1];b[0]++,b[b[0]+1]=b[b[0]]/DEPTH,b[b[0]]%=DEPTH);
    for (;!b[b[0]]&&b[0]>1;b[0]--);
}

void div(bignum_t c,bignum_t a,const bignum_t b){
    int h,l,m,i;
    memset((void*)c,0,sizeof(bignum_t));
    c[0]=(b[0]<a[0]+1)?(a[0]-b[0]+2):1;
    for (i=c[0];i;sub(a,b,c[i]=m,i-1),i--)
        for (h=DEPTH-1,l=0,m=(h+l+1)>>1;h>l;m=(h+l+1)>>1)
            if (comp(b,m,i-1,a)) h=m-1;
            else l=m;
    for (;!c[c[0]]&&c[0]>1;c[0]--);
    c[0]=c[0]>1?c[0]:1;
}

void div(bignum_t a,const int b,int& c){
    int i;
    for (c=0,i=a[0];i;c=c*DEPTH+a[i],a[i]=c/b,c%=b,i--);
    for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void sqrt(bignum_t b,bignum_t a){
    int h,l,m,i;
    memset((void*)b,0,sizeof(bignum_t));
    for (i=b[0]=(a[0]+1)>>1;i;sub(a,b,m,i-1),b[i]+=m,i--)
        for (h=DEPTH-1,l=0,b[i]=m=(h+l+1)>>1;h>l;b[i]=m=(h+l+1)>>1)
            if (comp(b,m,i-1,a)) h=m-1;
            else l=m;
    for (;!b[b[0]]&&b[0]>1;b[0]--);
    for (i=1;i<=b[0];b[i++]>>=1);
}

int length(const bignum_t a){
    int t,ret;
    for (ret=(a[0]-1)*DIGIT,t=a[a[0]];t;t/=10,ret++);
    return ret>0?ret:1;
}

int digit(const bignum_t a,const int b){
    int i,ret;
    for (ret=a[(b-1)/DIGIT+1],i=(b-1)%DIGIT;i;ret/=10,i--);
    return ret%10;
}

int zeronum(const bignum_t a){
    int ret,t;
```

```
    for (ret=0;!a[ret+1];ret++);
    for (t=a[ret+1],ret*=DIGIT;!(t%10);t/=10,ret++);
    return ret;
}

void comp(int* a,const int l,const int h,const int d){
    int i,j,t;
    for (i=l;i<=h;i++)
        for (t=i,j=2;t>1;j++)
            while (!(t%j))
                a[j]+=d,t/=j;
}

void convert(int* a,const int h,bignum_t b){
    int i,j,t=1;
    memset(b,0,sizeof(bignum_t));
    for (b[0]=b[1]=1,i=2;i<=h;i++)
        if (a[i])
            for (j=a[i];j;t*=i,j--)
                if (t*i>DEPTH)
                    mul(b,t),t=1;
    mul(b,t);
}

void combination(bignum_t a,int m,int n){
    int* t=new int[m+1];
    memset((void*)t,0,sizeof(int)*(m+1));
    comp(t,n+1,m,1);
    comp(t,2,m-n,-1);
    convert(t,m,a);
    delete []t;
}

void permutation(bignum_t a,int m,int n){
    int i,t=1;
    memset(a,0,sizeof(bignum_t));
    a[0]=a[1]=1;
    for (i=m-n+1;i<=m;t*=i++)
        if (t*i>DEPTH)
            mul(a,t),t=1;
    mul(a,t);
}

#define SGN(x) ((x)>0?1:((x)<0?-1:0))
#define ABS(x) ((x)>0?(x):-(x))

int read(bignum_t a,int &sgn,istream& is=cin){
```

```
    char str[MAX*DIGIT+2],ch,*buf;
    int i,j;
    memset((void*)a,0,sizeof(bignum_t));
    if (!(is>>str)) return 0;
    buf=str,sgn=1;
    if (*buf=='-') sgn=-1,buf++;
    for (a[0]=strlen(buf),i=a[0]/2-1;i>=0;i--)
        ch=buf[i],buf[i]=buf[a[0]-1-i],buf[a[0]-1-i]=ch;
    for (a[0]=(a[0]+DIGIT-1)/DIGIT,j=strlen(buf);j<a[0]*DIGIT;buf[j++]='0');
    for (i=1;i<=a[0];i++)
        for (a[i]=0,j=0;j<DIGIT;j++)
            a[i]=a[i]*10+buf[i*DIGIT-1-j]-'0';
    for (;!a[a[0]]&&a[0]>1;a[0]--);
    if (a[0]==1&&!a[1]) sgn=0;
    return 1;
}

struct bignum{
    bignum_t num;
    int sgn;
public:
inline bignum(){memset(num,0,sizeof(bignum_t));num[0]=1;sgn=0;}
inline int operator!(){return num[0]==1&&!num[1];}
inline bignum& operator=(const bignum& a){memcpy(num,a.num,sizeof(bignum_t));sgn=a.
sgn;return *this;}
inline bignum& operator=(const int a){memset(num,0,sizeof(bignum_t));num[0]=1;sgn=S
GN(a);add(num,sgn*a);return *this;};
inline bignum& operator+=(const bignum& a){if(sgn==a.sgn)add(num,a.num);else if(sgn
&&a.sgn){int ret=comp(num,a.num);if(ret>0)sub(num,a.num);else if(ret<0){bignum_t t;
    memcpy(t,num,sizeof(bignum_t));memcpy(num,a.num,sizeof(bignum_t));sub(num,t);sg
n=a.sgn;}else memset(num,0,sizeof(bignum_t)),num[0]=1,sgn=0;}else if(!sgn)memcpy(nu
m,a.num,sizeof(bignum_t)),sgn=a.sgn;return *this;}
inline bignum& operator+=(const int a){if(sgn*a>0)add(num,ABS(a));else if(sgn&&a){i
nt ret=comp(num,ABS(a));if(ret>0)sub(num,ABS(a));else if(ret<0){bignum_t t;
    memcpy(t,num,sizeof(bignum_t));memset(num,0,sizeof(bignum_t));num[0]=1;add(num,
ABS(a));sgn=-sgn;sub(num,t);}else memset(num,0,sizeof(bignum_t)),num[0]=1,sgn=0;}el
se if(!sgn)sgn=SGN(a),add(num,ABS(a));return *this;}
inline bignum operator+(const bignum& a){bignum ret;memcpy(ret.num,num,sizeof(bignu
m_t));ret.sgn=sgn;ret+=a;return ret;}
inline bignum operator+(const int a){bignum ret;memcpy(ret.num,num,sizeof(bignum_t)
);ret.sgn=sgn;ret+=a;return ret;}
inline bignum& operator-=(const bignum& a){if(sgn*a.sgn<0)add(num,a.num);else if(sg
n&&a.sgn){int ret=comp(num,a.num);if(ret>0)sub(num,a.num);else if(ret<0){bignum_t t
;
    memcpy(t,num,sizeof(bignum_t));memcpy(num,a.num,sizeof(bignum_t));sub(num,t);sg
n=-sgn;}else memset(num,0,sizeof(bignum_t)),num[0]=1,sgn=0;}else if(!sgn)add(num,a.
num),sgn=-a.sgn;return *this;}
```

```
inline bignum& operator-=(const int a){if(sgn*a<0)add(num,ABS(a));else if(sgn&&a){i
nt ret=comp(num,ABS(a));if(ret>0)sub(num,ABS(a));else if(ret<0){bignum_t t;
    memcpy(t,num,sizeof(bignum_t));memset(num,0,sizeof(bignum_t));num[0]=1;add(num,
ABS(a));sub(num,t);sgn=-sgn;}else memset(num,0,sizeof(bignum_t)),num[0]=1,sgn=0;}el
se if(!sgn)sgn=-SGN(a),add(num,ABS(a));return *this;}
inline bignum operator-(const bignum& a){bignum ret;memcpy(ret.num,num,sizeof(bignu
m_t));ret.sgn=sgn;ret-=a;return ret;}
inline bignum operator-(const int a){bignum ret;memcpy(ret.num,num,sizeof(bignum_t)
);ret.sgn=sgn;ret-=a;return ret;}
inline bignum& operator*=(const bignum& a){bignum_t t;mul(t,num,a.num);memcpy(num,t
,sizeof(bignum_t));sgn*=a.sgn;return *this;}
inline bignum& operator*=(const int a){mul(num,ABS(a));sgn*=SGN(a);return *this;}
inline bignum operator*(const bignum& a){bignum ret;mul(ret.num,num,a.num);ret.sgn=
sgn*a.sgn;return ret;}
inline bignum operator*(const int a){bignum ret;memcpy(ret.num,num,sizeof(bignum_t)
);mul(ret.num,ABS(a));ret.sgn=sgn*SGN(a);return ret;}
inline bignum& operator/=(const bignum& a){bignum_t t;div(t,num,a.num);memcpy(num,t
,sizeof(bignum_t));sgn=(num[0]==1&&!num[1])?0:sgn*a.sgn;return *this;}
inline bignum& operator/=(const int a){int t;div(num,ABS(a),t);sgn=(num[0]==1&&!num
[1])?0:sgn*SGN(a);return *this;}
inline bignum operator/(const bignum& a){bignum ret;bignum_t t;memcpy(t,num,sizeof(
bignum_t));div(ret.num,t,a.num);ret.sgn=(ret.num[0]==1&&!ret.num[1])?0:sgn*a.sgn;re
turn ret;}
inline bignum operator/(const int a){bignum ret;int t;memcpy(ret.num,num,sizeof(big
num_t));div(ret.num,ABS(a),t);ret.sgn=(ret.num[0]==1&&!ret.num[1])?0:sgn*SGN(a);ret
urn ret;}
inline bignum& operator%=(const bignum& a){bignum_t t;div(t,num,a.num);if (num[0]==
1&&!num[1])sgn=0;return *this;}
inline int operator%=(const int a){int t;div(num,ABS(a),t);memset(num,0,sizeof(bign
um_t));num[0]=1;add(num,t);return t;}
inline bignum operator%(const bignum& a){bignum ret;bignum_t t;memcpy(ret.num,num,s
izeof(bignum_t));div(t,ret.num,a.num);ret.sgn=(ret.num[0]==1&&!ret.num[1])?0:sgn;re
turn ret;}
inline int operator%(const int a){bignum ret;int t;memcpy(ret.num,num,sizeof(bignum
_t));div(ret.num,ABS(a),t);memset(ret.num,0,sizeof(bignum_t));ret.num[0]=1;add(ret.
num,t);return t;}
inline bignum& operator++(){*this+=1;return *this;}
inline bignum& operator--(){*this-=1;return *this;};
inline int operator>(const bignum& a){return sgn>0?(a.sgn>0?comp(num,a.num)>0:1):(s
gn<0?(a.sgn<0?comp(num,a.num)<0:0):a.sgn<0);}
inline int operator>(const int a){return sgn>0?(a>0?comp(num,a)>0:1):(sgn<0?(a<0?co
mp(num,-a)<0:0):a<0);}
inline int operator>=(const bignum& a){return sgn>0?(a.sgn>0?comp(num,a.num)>=0:1):
(sgn<0?(a.sgn<0?comp(num,a.num)<=0:0):a.sgn<=0);}
inline int operator>=(const int a){return sgn>0?(a>0?comp(num,a)>=0:1):(sgn<0?(a<0?
comp(num,-a)<=0:0):a<=0);}
inline int operator<(const bignum& a){return sgn<0?(a.sgn<0?comp(num,a.num)>0:1):(s
```

```
gn>0?(a.sgn>0?comp(num,a.num)<0:0):a.sgn>0);}
inline int operator<(const int a){return sgn<0?(a<0?comp(num,-a)>0:1):(sgn>0?(a>0?c
omp(num,a)<0:0):a>0);}
inline int operator<=(const bignum& a){return sgn<0?(a.sgn<0?comp(num,a.num)>=0:1):
(sgn>0?(a.sgn>0?comp(num,a.num)<=0:0):a.sgn>=0);}
inline int operator<=(const int a){return sgn<0?(a<0?comp(num,-a)>=0:1):(sgn>0?(a>0
?comp(num,a)<=0:0):a>=0);}
inline int operator==(const bignum& a){return (sgn==a.sgn)?!comp(num,a.num):0;}
inline int operator==(const int a){return (sgn*a>=0)?!comp(num,ABS(a)):0;}
inline int operator!=(const bignum& a){return (sgn==a.sgn)?comp(num,a.num):1;}
inline int operator!=(const int a){return (sgn*a>=0)?comp(num,ABS(a)):1;}
inline int operator[](const int a){return digit(num,a);}
friend inline istream& operator>>(istream& is,bignum& a){read(a.num,a.sgn,is);retur
n is;}
friend inline ostream& operator<<(ostream& os,const bignum& a){if(a.sgn<0)os<<'-';w
rite(a.num,os);return os;}
friend inline bignum sqrt(const bignum& a){bignum ret;bignum_t t;memcpy(t,a.num,siz
eof(bignum_t));sqrt(ret.num,t);ret.sgn=ret.num[0]!=1||ret.num[1];return ret;}
friend inline bignum sqrt(const bignum& a,bignum& b){bignum ret;memcpy(b.num,a.num,
sizeof(bignum_t));sqrt(ret.num,b.num);ret.sgn=ret.num[0]!=1||ret.num[1];b.sgn=b.num
[0]!=1||ret.num[1];return ret;}
inline int length(){return ::length(num);}
inline int zeronum(){return ::zeronum(num);}
inline bignum C(const int m,const int n){combination(num,m,n);sgn=1;return *this;}
inline bignum P(const int m,const int n){permutation(num,m,n);sgn=1;return *this;}
};
```

N 皇后构造解

```
//N 皇后构造解,n>=4

void even1(int n, int * p) {
    int i;
    for (i = 1; i <= n / 2; i++) {
        p[i - 1] = 2 * i;
    }
    for (i = n / 2 + 1; i <= n; i++) {
        p[i - 1] = 2 * i - n - 1;
    }
}

void even2(int n, int * p) {
    int i;
    for (i = 1; i <= n / 2; i++) {
        p[i - 1] = (2 * i + n / 2 - 3) % n + 1;
    }
    for (i = n / 2 + 1; i <= n; i++) {
        p[i - 1] = n - (2 (n - i + 1) + n / 2 - 3) % n;
```

```
    }
}

void generate(int, int);

void odd(int n, int * p) {
    generate(n - 1, p);
    p[n - 1] = n;
}

void generate(int n, int * p) {
    if (n & 1) {
        odd(n, p);
    } else if (n % 6 != 2) {
        even1(n, p);
    } else {
        even2(n, p);
    }
}
```

幻方构造

```
//幻方构造(L!=2)
const int MAXN = 100;

void dllb(int L, int si, int sj, int sn, int d[][MAXN]) {
    int n, i = 0, j = L / 2;
    for (n = 1; n <= L * L; n++) {
        d[i + si][j + sj] = n + sn;
        if (n % L) {
            i = (i) ? (i - 1): (L - 1);
            j = (j == L - 1) ? 0 : (j + 1);
        } else {
            i = (i == L - 1) ? 0 : (i + 1);
        }
    }
}

void magicOdd(int L, int d[][MAXN]) {
    dllb(L, 0, 0, 0, d);
}

void magic4k(int L, int d[][MAXN]) {
    int i, j;
    for (i = 0; i < L; i++) {
        for (j = 0; j < L; j++) {
            d[i][j] = ((i % 4 == 0 || i % 4 == 3) && (j % 4 == 0 || j % 4 == 3) ||
(i % 4 == 1 || i % 4 == 2) && (j % 4 == 1 || j % 4 == 2)) ? (L * L - (i * L + j)) :
```

```
  (i * L + j + 1);
            }
        }
    }

void magicOther(int L, int d[][MAXN]) {
    int i, j, t;
    dllb(L / 2, 0, 0, 0, d);
    dllb(L / 2, L / 2, L / 2, L * L / 4, d);
    dllb(L / 2, 0, L / 2, L * L / 2, d);
    dllb(L / 2, L / 2, 0, L * L / 4 * 3, d);
    for (i = 0; i < L / 2; i++) {
        for (j = 0; j < L / 4; j++) {
            if (i != L / 4 || j) {
                t = d[i][j];
                d[i][j] = d[i + L / 2][j];
                d[i + L / 2][j] = t;
            }
        }
    }
    t = d[L / 4][L / 4];
    d[L / 4][L / 4] = d[L / 4+L / 2][L / 4];
    d[L / 4+L / 2][L / 4] = t;
    for (i = 0; i < L / 2; i++) {
        for (j = L - L / 4+1; j < L; j++) {
            t = d[i][j];
            d[i][j] = d[i + L / 2][j];
            d[i + L / 2][j] = t;
        }
    }
}

void generate(int L, int d[][MAXN]) {
    if (L % 2) {
        magicOdd(L, d);
    } else if (L % 4 == 0) {
        magic4k(L, d);
    } else {
        magicOther(L, d);
    }
}
```

骰子

```
//Author: t__nt
//骰子的基本操作
//展开后对应的标号
//    3
```

```
//1 2 6 5
//    4
#include<cstdio>
//通过上表面和前面，得出右侧面
const int getFace[7][7]=//[upward][forward]
    //0 1 2 3 4 5 6
    {{0,0,0,0,0,0,0}     //0
    ,{0,0,3,5,2,4,0}     //1
    ,{0,4,0,1,6,0,3}     //2
    ,{0,2,6,0,0,1,5}     //3
    ,{0,5,1,0,0,6,2}     //4
    ,{0,3,0,6,1,0,4}     //5
    ,{0,0,4,2,5,3,0}};     //6
//对面的对应标号
const int oppo[7]={0,6,5,4,3,2,1};

class Dice{
public:
    int up,forward,right;

    Dice(){}

    Dice(int a,int b):up(a),forward(b){
        right=getFace[a][b];
    }

    Dice(int a,int b,int c):up(a),forward(b),right(c){}

    void goLeft(){
        int a=up,b=forward,c=right;
        up=c;
        forward=b;
        rigth=oppo[a];
    }

    void goRight(){
        int a=up,b=forward,c=right;
        up=oppo[c];
        forward=b;
        rigth=a;
    }

    void goUp(){
        int a=up,b=forward,c=right;
        up=b;
        forward=oppo[a];
        rigth=c;
```

```
    }

    void goDown(Pos & tmppos,Pos & pos){
        int a=up,b=forward,c=right;
        up=oppo[b];
        forward=a;
        rigth=c;
    }

};
```

第 k 元素

```
//一般可用 STL 的 kth_element()

//取第 k 个元素,k=0..n-1
//平均复杂度 O(n)
//注意 a[]中的顺序被改变
#define _cp(a, b) ((a) < (b))

template <class elemType>
elemType kthElement(int n, const elemType * a, int k) {
    elemType t, key;
    int left = 0, r = n - 1, i, j;
    while (left < r) {
        for (key = a[((i = left - 1) + (j = r + 1)) >> 1]; i < j;) {
            for (j--; _cp(key, a[j]); j--);
            for (i++; _cp(a[i], key); i++);
            if (i<j) {
                t = a[i];
                a[i] = a[j];
                a[j] = t;
            }
        }
        if (k > j) {
            left = j + 1;
        } else {
            r = j;
        }
    }
    return a[k];
}
```

最大子串匹配

```
//最大子串匹配,复杂度 O(mn)
//返回最大匹配值,传入两个串和串的长度,重载返回一个最大匹配
//注意做字符串匹配是串末的'\0'没有置!
//可更改元素类型,更换匹配函数和匹配价值函数
```

```
#include <cstring>
#include <algorithm>
using namespace std;

const int MAXN = 100;

#define _match(a, b) ((a) == (b))
#define _value(a, b) 1

template <class elemType>
int strMatch(int m, const elemType * a, int n, const elemType * b) {
    int match[MAXN + 1][MAXN + 1], i, j;
    memset(match, 0, sizeof(match));
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            match[i + 1][j + 1] = max(max(match[i][j + 1], match[i + 1][j]), (match
[i][j] + _value(a[i], b[i])) * _match(a[i], b[j]));
        }
    }
    return match[m][n];
}

template <class elemType>
int strMatch(int m, const elemType * a, int n, const elemType * b, elemType * ret)
{
    int match[MAXN + 1][MAXN + 1], last[MAXN + 1][MAXN + 1], i, j, t;
    memset(match, 0, sizeof(match));
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            match[i + 1][j + 1] = (match[i][j + 1] > match[i + 1][j] ? match[i][j +
 1] : match[i + 1][j]);
            last[i + 1][j + 1] = (match[i][j + 1] > match[i + 1][j] ? 3 : 1);
            if ((t = (match[i][j] + _value(a[i], b[i])) * _match(a[i], b[j]))> matc
h[i + 1][j + 1]) {
                match[i + 1][j + 1] = t;
                last[i + 1][j + 1] = 2;
            }
        }
    }
    for (; match[i][j]; i -= (last[t = i][j] > 1), j -= (last[t][j] < 3)) {
        ret[match[i][j] - 1] = (last[i][j] < 3 ? a[i - 1]: b[j - 1]);
    }
    return match[m][n];
}
```

最大子段和

```
//求最大子段和,复杂度O(n)
```

```
//传入串长 n 和内容 list[]
//返回最大子段和,重载返回子段位置(maxsum=list[start]+...+list[end])
//可更改元素类型
template <class elemType>
elemType maxsum(int n, const elemType * list) {
    elemType ret, sum = 0;
    int i;
    for (ret = list[i = 0]; i < n; i++) {
        sum = (sum > 0 ? sum : 0) + list[i];
        ret = (sum > ret ? sum : ret);
    }
    return ret;
}

template <class elemType>
elemType maxsum(int n, const elemType * list, int & start, int & end) {
    elemType ret, sum = 0;
    int s, i;
    for (ret = list[start = end = s = i = 0]; i < n; i++, s = (sum > 0 ? s : i)) {
        if ((sum = (sum > 0 ? sum : 0) + list[i])> ret) {
            ret = sum;
            start = s;
            end = i;
        }
    }
    return ret;
}
```

三维凸包

```
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>

class CH_vex;
class CH_edge;
class CH_face;

class CH
{
  friend class CH_edge;
  friend class CH_face;
 public:
  CH (double *v, int n_v);
  ~CH ();

  void compute       (void);

  int get_n_v        (void);//get the number of vertices on the surface
  int get_n_faces    (void);//get the number of triangles on the surface
  int get_convex_hull (double **v, int *n_v, int **faces, int *n_faces);//get  the convex hull

 private:
  void add_vex       (CH_vex *v);
  void add_edge      (CH_edge *e);
  void add_face      (CH_face *f);
  void del_vex       (CH_vex *v);
```

```
  void del_edge       (CH_edge *e);
  void del_face       (CH_face *f);
  int are_collinear   (CH_vex *v1, CH_vex *v2, CH_vex *v3);
  int volume_sign     (CH_face *f, CH_vex *v);
  int add_one         (CH_vex *v);
  void clean_up       (CH_vex *vnext);
  void clean_edges    (void);
  void clean_faces    (void);
  void clean_v        (CH_vex *vnext);
  int double_triangle (void);
  int construct_hull  (void);
  int get_vex_index   (CH_vex *v);
  CH_vex  *v;
  CH_edge *edges;
  CH_face *faces;
};

class CH_vex
{
  friend class CH;
  friend class CH_face;
 public:
  CH_vex () {};
  CH_vex (double x, double y, double z);

 private:
  double        pt[3];
  CH_edge   *duplicate;
  short     on_hull;
  short     processed;
  CH_vex *prev, *next;
};

class CH_edge
{
  friend class CH;
  friend class CH_face;
 public:
  CH_edge (CH *hull3D);

 private:
  CH_face   *adjf[2];
  CH_vex    *endp[2];
  CH_face   *new_face;
  int       to_del;
  CH_edge   *prev, *next;
};

class CH_face
{
  friend class CH;
 public:
  CH_face (CH *hull3D, CH_vex *v1, CH_vex *v2, CH_vex *v3, CH_face *f);
  CH_face (CH *hull3D, CH_edge *e, CH_vex *v);

 private:
  void make_ccw (CH_edge *e, CH_vex *v);

 private:
  CH_edge   *edges[3];
  CH_vex    *v[3];
  int       visible;
  CH_face   *prev, *next;
};

#define SWAP(t,x,y) { t = x; x = y; y = t; }
#define ADD( head, p )  if ( head )  { \
       p->next = head; \
       p->prev = head->prev; \
       head->prev = p; \
       p->prev->next = p; \
```

```
    } \
    else { \
        head = p; \
        head->next = head->prev = p; \
    }

#define del( head, p ) if ( head ) { \
        if ( head == head->next ) \
        head = NULL; \
        else if ( p == head ) \
        head = head->next; \
        p->next->prev = p->prev; \
        p->prev->next = p->next; \
        del p; \
    }

CH::CH (double *v, int n)
{
    int i;//assert v);

    v = NULL;
    for (i=0; i<n; i++)
    add_vex (new CH_vex (v[3*i], v[3*i+1], v[3*i+2]));
    edges = NULL;
    faces = NULL;
}

CH::~CH ()
{
    CH_vex *v, *vt;
    CH_edge *e, *et;
    CH_face *f, *ft;

    v = v;
    do {
        vt = v;
        v = v->next;
        del_vex (vt);
    } while (v->next != v);
    del_vex (v);
    e = edges;
    do {
        et = e;
        e = e->next;
        del_edge (et);
    } while (edges->next != edges);
    del_edge (edges);
    f = faces;
    do {
        ft = f;
        f = f->next;
        del_face (ft);
    } while (faces->next != faces);
    del_face (faces);
}

void CH::add_vex    (CH_vex *v) { ADD (v, v); }
void CH::add_edge     (CH_edge *e)   { ADD (edges, e); }
void CH::add_face     (CH_face *f)   { ADD (faces, f); }
void CH::del_vex (CH_vex *v) { del (v, v); }
void CH::del_edge   (CH_edge *e)   { del (edges, e); }
void CH::del_face   (CH_face *f)   { del (faces, f); }

int CH::get_n_v (){
    CH_vex *v = v;
    int n=0;
    if (!v)
    return 0;
    else
    do { v = v->next; n++; } while (v != v);
```

```
    return n;
}

int CH::get_n_faces (){
    CH_face *f = faces;
    int n=0;
    if (!faces)
    return 0;
    else
    do { f = f->next; n++; } while (f != faces);
    return n;
}

int CH::get_vex_index (CH_vex *v){
    CH_vex *v_walk = v;
    int index = 0;
    do {
        if (v_walk == v)
        return index;
        index++;
        v_walk = v_walk->next;
    } while (v_walk != v);
    return -1;
}

int CH::get_convex_hull (double **v, int *nv, int **f, int *nf){
    *nv = get_n_v ();
    *nf = get_n_faces ();

    double *cv_v = (double*)malloc(3*(*nv)*sizeof(double));
    int   *cv_faces    = (int*)malloc(3*(*nf)*sizeof(int));
    if (!cv_v || !cv_faces)
    {
        v = NULL; f = NULL;
        *nv = *nf = 0;
        return 0;
    }
    CH_vex *v_walk=v;
    int i=0;
    do {
        cv_v[3*i]   = v_walk->pt[0];
        cv_v[3*i+1] = v_walk->pt[1];
        cv_v[3*i+2] = v_walk->pt[2];
        v_walk = v_walk->next;
        i++;
    } while (v_walk != v);
    i = 0;
    CH_face *f_walk = faces;
    do {
        cv_faces[3*i]   = get_vex_index (f_walk->v[0]);
        cv_faces[3*i+1] = get_vex_index (f_walk->v[1]);
        cv_faces[3*i+2] = get_vex_index (f_walk->v[2]);
        f_walk = f_walk->next;
        i++;
    } while (f_walk != faces);
    *v = cv_v;
    *f = cv_faces;
    return 1;
}

void CH::compute (){
    double_triangle ();
    construct_hull ();
}

int CH::double_triangle(){
    CH_vex *v0, *v1, *v2, *v3;
    int vol;

    v0 = v;
    while (are_collinear (v0, v0->next, v0->next->next))
    if ( (v0=v0->next) == v)    {//All the v are collinear
```

```
        return 1;
    }
    v1 = v0->next;
    v2 = v1->next;
    v0->processed = 1;
    v1->processed = 1;
    v2->processed = 1;
    CH_face *f0, *f1 = NULL;
    f0 = new CH_face (this, v0, v1, v2, f1);
    f1 = new CH_face (this, v2, v1, v0, f0);
    f0->edges[0]->adjf[1] = f1;
    f0->edges[1]->adjf[1] = f1;
    f0->edges[2]->adjf[1] = f1;
    f1->edges[0]->adjf[1] = f0;
    f1->edges[1]->adjf[1] = f0;
    f1->edges[2]->adjf[1] = f0;
    v3 = v2->next;
    vol = volume_sign (f0, v3);
    while (!vol)
    {
        if ( (v3=v3->next) == v0){//All the v are coplanar
            return 1;
        }
        vol = volume_sign (f0, v3);
    }
    v = v3;
    return 0;
}

int CH::construct_hull(){
    CH_vex *v, *vnext;

    v = v;
    do {
        vnext = v->next;
        if (!v->processed){
            v->processed = 1;
            add_one (v);
            clean_up (vnext);
        }
        v = vnext;
    } while (v != v);
    return 0;
}

int CH::are_collinear (CH_vex *v1, CH_vex *v2, CH_vex *v3){
    return
    (( v3->pt[2] - v1->pt[2] ) * ( v2->pt[1] - v1->pt[1] ) -
    ( v2->pt[2] - v1->pt[2] ) * ( v3->pt[1] - v1->pt[1] ) == 0
    && ( v2->pt[2] - v1->pt[2] ) * ( v3->pt[0] - v1->pt[0] ) -
    ( v2->pt[0] - v1->pt[0] ) * ( v3->pt[2] - v1->pt[2] ) == 0
    && ( v2->pt[0] - v1->pt[0] ) * ( v3->pt[1] - v1->pt[1] ) -
    ( v2->pt[1] - v1->pt[1] ) * ( v3->pt[0] - v1->pt[0] ) == 0);
}

int CH::volume_sign (CH_face *f, CH_vex *v){
    double ax, ay, az, bx, by, bz, cx, cy, cz;
    double vol;

    ax = f->v[0]->pt[0] - v->pt[0];
    ay = f->v[0]->pt[1] - v->pt[1];
    az = f->v[0]->pt[2] - v->pt[2];
    bx = f->v[1]->pt[0] - v->pt[0];
    by = f->v[1]->pt[1] - v->pt[1];
    bz = f->v[1]->pt[2] - v->pt[2];
    cx = f->v[2]->pt[0] - v->pt[0];
    cy = f->v[2]->pt[1] - v->pt[1];
    cz = f->v[2]->pt[2] - v->pt[2];

    vol =   ax * (by*cz - bz*cy)
    + ay * (bz*cx - bx*cz)
    + az * (bx*cy - by*cx);
```

```
    if      ( vol >  0.0 ) return  1;
    else if ( vol <  0.0 ) return -1;
    else                   return  0;
}

int CH::add_one (CH_vex *v){
    CH_face *f;
    CH_edge *e, *temp;
    int vol;
    int vis = 0;

    f = faces;
    do {
        vol = volume_sign (f, v);
        if (vol<0){
            f->visible = 1;
            vis = 1;
        }
        f = f->next;
    } while (f != faces);
    if (!vis){
        v->on_hull = 0;
        return 0;
    }
    e = edges;
    do {
        temp = e->next;
        if (e->adjf[0]->visible && e->adjf[1]->visible)
            e->to_del = 1;
        else if (e->adjf[0]->visible || e->adjf[1]->visible)
            e->new_face = new CH_face (this, e, v);
        e = temp;
    } while (e != edges);
    return 1;
}

void CH::clean_up (CH_vex *vnext){
    clean_edges ();
    clean_faces ();
    clean_v (vnext);
}

void CH::clean_edges (void){
    CH_edge *e, *t;

    e = edges;
    do {
        if (e->new_face){
            if (e->adjf[0]->visible) e->adjf[0] = e->new_face;
            else                     e->adjf[1] = e->new_face;
            e->new_face = NULL;
        }
        e = e->next;
    } while (e != edges);
    while (edges && edges->to_del){
        e = edges;
        del_edge (e);
    }
    e = edges->next;
    do {
        if (e->to_del){
            t = e;
            e = e->next;
            del_edge (t);
        }else
            e = e->next;
    } while (e != edges);
}

void CH::clean_faces(){
    CH_face *f, *t;
```

```
    while (faces && faces->visible)
    {
        f = faces;
        del_face (f);
    }
    f = faces->next;
    do {
        if (f->visible)
        {
            t = f;
            f = f->next;
            del_face (t);
        }
        else
        f = f->next;
    } while (f != faces);
}

void CH::clean_v (CH_vex *vnext){
    CH_edge *e;
    CH_vex *v, *t;

    e = edges;
    do {
        e->endp[0]->on_hull = e->endp[1]->on_hull = 1;
        e = e->next;
    } while (e != edges);
    while (v && v->processed && !v->on_hull){
        if (v == vnext)
        vnext = v->next;
        v = v;
        del_vex (v);
    }
    v = v->next;
    do {
        if (v->processed && !v->on_hull)
        {
            t = v;
            v = v->next;
            del_vex (t);
        }
        else
        v = v->next;
    } while (v != v);
    v = v;
    do {
        v->duplicate = NULL;
        v->on_hull = 0;
        v = v->next;
    } while (v != v);
}

CH_vex::CH_vex (double x, double y, double z){
    pt[0] = x;  pt[1] = y;  pt[2] = z;
    duplicate = NULL;
    on_hull   = 0;
    processed = 0;
}

CH_edge::CH_edge (CH *hull3D){
    adjf[0]  = adjf[1]  = new_face = NULL;
    endp[0] = endp[1] = NULL;
    to_del = 0;
    hull3D->add_edge (this);
}

CH_face::CH_face (CH *hull3D, CH_vex *v1, CH_vex *v2, CH_vex *v3, CH_face *f){
    CH_edge *e0, *e1, *e2;

    if(!f){
        e0 = new CH_edge (hull3D);
```

```
        e1 = new CH_edge (hull3D);
        e2 = new CH_edge (hull3D);
    }else{
        e0 = f->edges[2];
        e1 = f->edges[1];
        e2 = f->edges[0];
    }
    e0->endp[0] = v1;       e0->endp[1] = v2;
    e1->endp[0] = v2;       e1->endp[1] = v3;
    e2->endp[0] = v3;       e2->endp[1] = v1;
    edges[0]    = e0;    edges[1]    = e1;    edges[2]    = e2;
    v[0] = v1;    v[1] = v2;    v[2] = v3;
    visible = 0;
    e0->adjf[0] = e1->adjf[0] = e2->adjf[0] = this;
    hull3D->add_face (this);
}

CH_face::CH_face (CH *hull3D, CH_edge *e, CH_vex *v){
    CH_edge *new_edges[2];
    int i,j;

    for (i=0; i<2; ++i)
    if (!(new_edges[i] = e->endp[i]->duplicate)){
        new_edges[i] = new CH_edge (hull3D);
        new_edges[i]->endp[0] = e->endp[i];
        new_edges[i]->endp[1] = v;
        e->endp[i]->duplicate = new_edges[i];
    }
    edges[0] = e;
    edges[1] = new_edges[0];
    edges[2] = new_edges[1];
    visible = 0;
    make_ccw (e, v);
    for (i=0; i<2; ++i)
    for (j=0; j<2; ++j)
    if (!new_edges[i]->adjf[j]){
        new_edges[i]->adjf[j] = this;
        break;
    }
    hull3D->add_face (this);
}

void CH_face::make_ccw (CH_edge *e, CH_vex *v){
    CH_face *fv;
    int i;
    CH_edge *s;

    if (e->adjf[0]->visible) fv = e->adjf[0];
    else                     fv = e->adjf[1];
    for (i=0; fv->v[i] != e->endp[0]; ++i);
    if (fv->v[(i+1)%3] != e->endp[1]){
        v[0] = e->endp[1];
        v[1] = e->endp[0];
    }else{
        v[0] = e->endp[0];
        v[1] = e->endp[1];
        SWAP (s, edges[1], edges[2]);
    }
    v[2] = v;
}

double f[4096];

int main()
{
    int testnum, n;

    scanf("%d",&testnum);
    for(int test = 1;test <= testnum;test++){
        scanf("%d",&n);
        for(int i = 0;i < n;i++){
            scanf("%lf%lf%lf",&f[i * 3],&f[i * 3 + 1],&f[i * 3 + 2]);
```

```
        }
        CH D = CH(f,n);
        D.compute();
        printf("Number of triangles on the surface: %d\n",D.get_n_faces());
    }
    return 0;
}
```

例程_Linyue

这一部分未能整理成模块-_-b

第K短路

```
// 第K短路A*算法，edge2为星形图的存储方法，和最大流以及费用流的存储一样
#include <iostream>
#include <cstdio>
#include <queue>

using namespace std;
const int inf = 1000000000;
const int maxQ = 100000;
const int N = 1000000;
const int maxn = 2000;
struct Tedge{
    int to, len, next;
}edge1[N], edge2[N];
int dis[maxn], start1[maxn], start2[maxn];
bool avai[maxn];
int s, t, k, n, m, nowe1, nowe2;
int Q[maxQ];
struct node{
    int u, l, h;
    node(int uu = 0, int ll = 0, int hh = 0) : u(uu), l(ll), h(hh){}
    bool operator<(const node &p)const{
        return h > p.h;
    }
};
priority_queue<node> g;

void AddEdge1(int u, int v, int w){
    edge1[nowe1].to = v;
    edge1[nowe1].len = w;
    edge1[nowe1].next = start1[u];
    start1[u] = nowe1 ++;
}

void AddEdge2(int u, int v, int w){
    edge2[nowe2].to = v;
    edge2[nowe2].len = w;
    edge2[nowe2].next = start2[u];
    start2[u] = nowe2 ++;
}

void SPFA(){
```

```
    int open, closed, u;
    for (int i = 0; i < n; ++i) dis[i] = inf;
    dis[t] = 0;
    memset(avai, 0, sizeof(avai));
    open = -1; closed = 0;
    Q[0] = t; avai[t] = true;
    while (open != closed){
        open = (open + 1) % maxQ;
        u = Q[open];
        int tmp = start2[u];
        while (tmp != -1){
            int v = edge2[tmp].to;
            int len = edge2[tmp].len;
            if (dis[u] + len < dis[v]){
                dis[v] = dis[u] + len;
                if (avai[v] == false){
                    closed = (closed + 1) % maxQ;
                    Q[closed] = v;
                    avai[v] = true;
                }
            }
            tmp = edge2[tmp].next;
        }
        avai[u] = false;
    }
}

void Astar(){
    int c[maxn];
    if (dis[s] == inf){
        printf("-1\n");
        return;
    }
    memset(c, 0, sizeof(c));
    if (s == t) ++k;
    node now;
    now.u = s;
    now.l = 0;
    now.h = dis[s];
    g.push(now);
    while (!g.empty()){
        now = g.top();
        g.pop();
        c[now.u] ++;
        if (c[t] == k){
            printf("%d\n",now.l);
            return;
```

```
        }
        if (c[now.u] > k) continue;
        int tmp = start1[now.u];
        while (tmp != -1){
            int v = edge1[tmp].to;
            int len = edge1[tmp].len;
            tmp = edge1[tmp].next;
            node temp;
            temp.u = v;
            temp.l = now.l + len;
            temp.h = temp.l + dis[v];
            if (dis[v] == inf) continue;
            g.push(temp);
        }
    }
    printf("-1\n");
}

int main(){
    int u, v, w;
    nowe1 = nowe2 = 0;
    memset(start1, -1, sizeof(start1));
    memset(start2, -1, sizeof(start2));
    scanf("%d%d",&n, &m);
    for (int i = 0; i < m; ++i){
        scanf("%d%d%d",&u, &v, &w);
        --u, --v;
        AddEdge1(u, v, w);
        AddEdge2(v, u, w);
    }
    scanf("%d%d%d",&s, &t, &k);
    --s, --t;
    SPFA();
    Astar();
    return 0;
}
```

费用流

```
/*
Method:最小费用最大流，可处理重边，用 SPFA 找最短路。
Author: LCLL
*/
#include <iostream>
#include <cstdio>
#include <string>
#include <algorithm>
```

```
using namespace std;

#define LONG long long
#define maxpnt 350 //点数
#define maxe 4000 //边数
#define maxQ 36000
int a, b, c, n, s, tot;
LONG ans;

struct TEdge {
    int pnt, capa, next;
    LONG cost;
};
//nowe 必须每次置 0，start 要初始为-1,source 和 terminal 要自己赋值
int source, terminal, nowe, open, closed, nn;
bool avai[maxpnt];
LONG dis[maxpnt];
int start[maxpnt], last[maxpnt];
int Q[maxQ];
TEdge edges[maxe << 1];

//添加有向边 n1->n2,容量为 n3,费用 n4
void AddEdge(int n1, int n2, int n3, int n4) {
    edges[nowe].pnt = n2;
    edges[nowe].capa = n3;
    edges[nowe].cost = n4;
    edges[nowe].next = start[n1];
    start[n1] = nowe++;

    edges[nowe].pnt = n1;
    edges[nowe].capa = 0;
    edges[nowe].cost = -n4;
    edges[nowe].next = start[n2];
    start[n2] = nowe++;
}

void process()
{
    ans = 0; //最小费用
    while (true) {
        memset(avai, true, sizeof(avai[0]) * terminal + 1);
        for (int i = 0; i <= terminal; ++i) dis[i] = -1;
        dis[source] = 0;
        open = -1; closed = 0;
        Q[0] = source; avai[source] = false;
        int now;
        while (open != closed) {
```

```
            open = (open + 1) % maxQ;
            now = Q[open];
            int i = start[now];
            while (i != -1) {
                if (edges[i].capa && (dis[now] + edges[i].cost < dis[edges[i].pnt]
|| (dis[edges[i].pnt] == -1))) {
                    dis[edges[i].pnt] = dis[now] + edges[i].cost;
                    last[edges[i].pnt] = i;
                    if (avai[edges[i].pnt]) {
                        avai[edges[i].pnt] = false;
                        closed = (closed + 1) % maxQ;
                        Q[closed] = edges[i].pnt;
                    }
                }
                i = edges[i].next;
            }
            avai[now] = true;
        }

        if (dis[terminal] == -1) break;

        now = terminal;
        while (now != source) {
            --edges[last[now]].capa;
         ans += edges[last[now]].cost;
            last[now] ^= 1;
            ++edges[last[now]].capa;
            now = edges[last[now]].pnt;
        }
    }
}
```

快速傅立叶变换

```
//快速傅立叶变换
const int MAXN = 524288;
const double PI = 2 * acos(0.0);
const int UNIT = 4;
const int B = 10000;

struct BigN {
   int d[MAXN];
   int len;
};

struct cplx {
   double r, i;
};
```

```
cplx w[MAXN / 2];
cplx fft1[MAXN], fft2[MAXN], coef[MAXN];
int nfft;
char buf[1000001];
BigN a, b;

void
Init(void)
{
   double step = 2 * PI / MAXN;
   for ( int i = 0; 2 * i < MAXN; i++ ) {
      double a = i * step;
      w[i].r = cos(a);
      w[i].i = sin(a);
   }
}

void
RFFT(cplx a[], cplx fft[], int n, int step, int s)
{
   if ( n == 1 ) {
      fft[0] = a[0];
      return;
   }
   int n2 = n / 2;
   cplx *f1 = fft;
   cplx *f2 = fft + n2;
   RFFT(a, f1, n2, step * 2, s);
   RFFT(a + step, f2, n2, step * 2, s);
   int stepw = MAXN / n;
   int p = 0;
   for ( int i = 0; i < n2; i++ ) {
      double tr = w[p].r * f2[i].r - s * w[p].i * f2[i].i;
      double ti = w[p].r * f2[i].i + s * w[p].i * f2[i].r;
      f2[i].r = f1[i].r - tr;
      f2[i].i = f1[i].i - ti;
      f1[i].r += tr;
      f1[i].i += ti;
      p += stepw;
   }
}

void
FFT(int a[], cplx fft[])
{
   for ( int i = 0; i < nfft; i++ ) {
```

```
        coef[i].r = a[i];
        coef[i].i = 0;
    }
    RFFT(coef, fft, nfft, 1, 1);
}

void
Print(const BigN &n, char *str = "\n")
{
    printf("%d", n.d[n.len - 1]);
    for ( int i = n.len - 2; i >= 0; i-- )
        printf("%0*d", UNIT, n.d[i]);
    printf("%s", str);
}

void
Set(BigN &n, char *str)
{
    memset(&n, 0, sizeof n);
    int len = strlen(str);
    int s = 0;
    while ( s < len - 1 && str[s] == 0 ) ;
    int num = 0;
    int wt = 1;
    for ( int i = len - 1; i >= s; i-- ) {
        num += (str[i] - '0') * wt;
        if ( (len - i) % 4 == 0 ) {
            n.d[n.len++] = num;
            num = 0;
            wt = 1;
        } else
            wt *= 10;
    }
    if ( num ) n.d[n.len++] = num;
}

void
Mul(BigN &a, BigN &b)
{
    nfft = 2;
    while ( nfft < a.len + b.len ) nfft *= 2;
    FFT(a.d, fft1);
    FFT(b.d, fft2);
    for ( int i = 0; i < nfft; i++ ) {
        coef[i].r = fft1[i].r * fft2[i].r - fft1[i].i * fft2[i].i;
        coef[i].i = fft1[i].r * fft2[i].i + fft1[i].i * fft2[i].r;
    }
```

```
    RFFT(coef, fft1, nfft, 1, -1);
    double invn = (double)1 / nfft;
    a.len += b.len;
    double c = 0;
    for ( int i = 0; i < a.len; i++ ) {
        double d = floor(fft1[i].r * invn + 0.5) + c;
        c = floor(d / B);
        a.d[i] = (int)(d - c * B);
    }
    while ( a.len > 1 && a.d[a.len - 1] == 0 )
        --a.len;
    Print(a);
}

int
main()
{
    Init();
    int T;
    scanf("%d", &T);
    while ( T-- > 0 ) {
        scanf("%s", buf);
        Set(a, buf);
        scanf("%s", buf);
        Set(b, buf);
        Mul(a, b);
    }
}
```

判断凹凸

```
/*
判断是凹集还是凸集。可信度未知
语法：result=convex(Point *p,int n);
参数：
*p：封闭曲线顶点数组
n：封闭曲线顶点个数
返回值：1：凸集；-1：凹集；0：曲线不符合要求无法计算
注意：
默认曲线为简单曲线：无交叉、无圈
*/

        typedef struct {
            double x,y;
        } Point;
        int convex(Point *p,int n)
        {
            int i,j,k;
```

```
        int flag = 0;
        double z;

        if (n < 3)
            return(0);
        for (i=0;i<n;i++) {
            j = (i + 1) % n;
            k = (i + 2) % n;
            z = (p[j].x - p[i].x) * (p[k].y - p[j].y);
            z -= (p[j].y - p[i].y) * (p[k].x - p[j].x);
            if (z < 0)
                flag |= 1;
            else if (z > 0)
                flag |= 2;
            if (flag == 3)
                return -1; //CONCAVE
        }
        if (flag != 0)
            return 1; //CONVEX
        else
        return 0;
    }
```

平面图最大流

```
// Input: N and M, number of cities and edges. Next N lines give coordinates of the
 cities,
// with minimum x and maximum x being s and t respectively. Following M lines give
the
// undirected edges between cities.

const double EPS = 1E-10;
const double PI = 2 * acos(0.0);
const int N = 100000;
const int E = N * 6;

struct Edge {
   int v, next;
   double c;
};

struct Ent {
   double a;
   int e;
   bool operator<(const Ent &b) const {
       return a < b.a;
   }
};
```

```
struct Ent2 {
   double d;
   int v;
   bool operator<(const Ent2 &b) const {
      return d > b.d;
   }
};

int G[N], ne, s, t, G2[N * 2], ne2, sd, td;
int n, m, nf;
int X[N], Y[N];
Edge edge[E], edge2[E];
int next[E];
Ent A[N];
int f[E];
Ent2 heap[E];
int nheap;
bool vis[N * 2];
double dist[N * 2];

void
Insert(int a, int b, double c, int g[], Edge e[], int &ne)
{
   e[ne].v = b;
   e[ne].c = c;
   e[ne].next = g[a];
   g[a] = ne++;
}

void
get_embedding(int u)
{
   int k = 0;
   for ( int i = G[u]; i != -1; i = edge[i].next ) {
      int v = edge[i].v;
      A[k].e = i;
      A[k].a = atan2(Y[v] - Y[u], X[v] - X[u]);
      ++k;
   }
   if ( u == s ) A[0].a = PI * 3 / 4;
   else if ( u == t ) A[0].a = PI / 4;
   sort(A, A + k);
   for ( int i = 0; i < k - 1; i++ )
      next[A[i].e] = A[i + 1].e;
   next[A[k - 1].e] = A[0].e;
}
```

```
void
find_face()
{
   for ( int i = 0; i < n; i++ )
      get_embedding(i);
   nf = 0;
   memset(f, -1, sizeof(*f) * ne);
   for ( int e = 0; e < ne; e++ ) {
      if ( f[e] >= 0 ) continue;
      while ( f[e] == -1 ) {
         f[e] = nf;
         e = next[e ^ 1];
      }
      ++nf;
   }
}

void
init()
{
   find_face();
   sd = f[ne - 2];
   td = f[ne - 1];
   memset(G2, -1, sizeof(*G2) * nf);
   for ( int i = 0; i < ne - 2; i++ ) {
      int u = f[i], v = f[i ^ 1];
      Insert(u, v, edge[i].c, G2, edge2, ne2);
   }
}

void
Solve()
{
   init();
   for ( int i = 0; i < nf; i++ ) dist[i] = 1E100;
   heap[0].v = sd; heap[0].d = 0; nheap = 1;
   while ( nheap ) {
      int u = heap[0].v;
      double d = heap[0].d;
      pop_heap(heap, heap + nheap--);
      if ( u == td ) break;
      if ( vis[u] ) continue;
      vis[u] = 1;
      for ( int i = G2[u]; i != -1; i = edge2[i].next ) {
         int v = edge2[i].v;
         double nd = d + edge2[i].c;
```

```
            if ( nd < dist[v] - EPS ) {
                dist[v] = nd;
                heap[nheap].v = v;
                heap[nheap].d = nd;
                push_heap(heap, heap + ++nheap);
            }
        }
    }
    printf("%.4f\n", dist[td]);
}

int
main()
{
    scanf("%d %d", &n, &m);
    memset(G, -1, sizeof(*G) * n);
    for ( int i = 0; i < n; i++ )
        scanf("%d %d", X + i, Y + i);
    s = min_element(X, X + n) - X;
    t = max_element(X, X + n) - X;
    for ( int i = 0; i < m; i++ ) {
        int a, b;
        double c;
        scanf("%d %d %lf", &a, &b, &c);
        Insert(a, b, c, G, edge, ne);
        Insert(b, a, c, G, edge, ne);
    }
    Insert(s, t, 0, G, edge, ne);
    Insert(t, s, 0, G, edge, ne);
    Solve();
}
```

网络流

```
// 注意事项和费用流一样。start 要初赋值-1，nowe=0。要赋值 terminal 和 source。ans 为最终结果。

/*
Method: 最大网络流 Dinic 算法,可处理重边。
可靠程度: 5 颗星
*/
#include <cstdio>
#include <algorithm>
using namespace std;
const int maxpnt = 60000;
const int maxe = 200000;
const int oo = 2147483647;
struct TEdge {
    int pnt, capa, next;
```

```
};
int n, m, nowe, source, terminal, ans, open, closed;
bool avai[maxpnt];
int last[maxpnt], start[maxpnt], aug[maxpnt], Q[maxpnt], level[maxpnt];
int startnow[maxpnt];
TEdge edges[maxe << 1];
inline void AddEdge(int n1, int n2, int n3) {
    edges[nowe].pnt = n2;
    edges[nowe].capa = n3;
    edges[nowe].next = start[n1];
    start[n1] = nowe++;
    edges[nowe].pnt = n1;
    edges[nowe].capa = 0;
    edges[nowe].next = start[n2];
    start[n2] = nowe++;
}
inline void init() { //读入部分，这里就不删除了。
    source = 0; terminal = n + m + 1;
    nowe = 0;
    memset(start, 255, sizeof(start[0]) * (terminal + 1));
    int n1, n2, n3;
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &n1);
        AddEdge(m + i, terminal, n1);
    }
    ans = 0;
    for (int i = 1; i <= m; ++i) {
        scanf("%d%d%d", &n1, &n2, &n3);
        ans += n3;
        AddEdge(source, i, n3);
        AddEdge(i, n1 + m, oo);
        AddEdge(i, n2 + m, oo);
    }
}
inline void GetLevel() {
    memset(level, 255, sizeof(level[0]) * (terminal + 1));
    level[source] = 0;
    Q[0] = source;
    open = -1; closed = 0;
    int now;
    while (open < closed && level[terminal] == -1) {
        now = Q[++open];
        int i = start[now];
        while (i != -1) {
            if (edges[i].capa && level[edges[i].pnt] == -1) {
                level[edges[i].pnt] = level[now] + 1;
                Q[++closed] = edges[i].pnt;
```

```
            }
            i = edges[i].next;
        }
    }
}
inline void process() {
    while (true) {
        GetLevel();
        if (level[terminal] == -1) break;
        memset(aug, 0, sizeof(aug[0]) * (terminal + 1));
        aug[source] = oo;
        memset(avai, true, sizeof(avai[0]) * (terminal + 1));
        memcpy(startnow, start, sizeof(start[0]) * (terminal + 1));
        int now = source;
        while (true) {
            int i = startnow[now];
            while (i != -1) {
                if (edges[i].capa && level[now] + 1 == level[edges[i].pnt]
        && avai[edges[i].pnt]) {
                    last[edges[i].pnt] = i;
                    aug[edges[i].pnt] = min(edges[i].capa, aug[now]);
                    startnow[now] = i;
                    now = edges[i].pnt;
                    break;
                }
                i = edges[i].next;
            }
            if (i == -1) {
                avai[now] = false;
                if (now == source) break;
                now = edges[last[now] ^ 1].pnt;
            }
            else
                if (now == terminal) {
                    int delta = aug[terminal];
                    ans += delta;
                    int i = terminal, tmp;
                    while (i != source) {
                        aug[i] -= delta;
                        edges[last[i]].capa -= delta;
                        tmp = last[i] ^ 1;
                        if (aug[i] == 0) now = edges[tmp].pnt;
                        edges[tmp].capa += delta;
                        i = edges[tmp].pnt;
                    }
                }
        }
```

```
        }
    }
```

支配集

```
// PKU Agents, 要找|U|<=3 的支配集
const int V = 500;

int nvert, ne;
int dom[3];
int G[V][V], deg[V];
int domcol[3];
int list[V][2], nlist[V];
bool has[3];
int edge[V * 2][V * 2];
int nedge[V * 2];
bool vis[V];
int col[V];

inline int
InDom(int v, int n)
{
    for ( int i = 0; i < n; i++ )
        if ( dom[i] == v ) return i;
    return -1;
}

inline bool
Dominated(int v, int n)
{
    if ( InDom(v, n) >= 0 ) return 1;
    for ( int i = 0; i < deg[v]; i++ )
        if ( InDom(G[v][i], n) >= 0 ) return 1;
    return 0;
}

int
FindDom(int v, int n)
{
    if ( n > 3 ) return n;
    for ( int i = v; i < nvert; i++ ) {
        if ( ! Dominated(i, n) ) {
            dom[n] = i;
            int s = FindDom(i + 1, n + 1);
            if ( s <= 3 ) return s;
            for ( int j = 0; j < deg[i]; j++ ) {
                dom[n] = G[i][j];
                s = FindDom(i + 1, n + 1);
```

```
            if ( s <= 3 ) return s;
         }
         return 4;
      }
   }
   return n;
}

inline void
Add(int i, int a, int j, int b)
{
   int c = i * 2 + a, d = j * 2 + b;
   edge[c][nedge[c]++] = d ^ 1;
   edge[d][nedge[d]++] = c ^ 1;
}

void
Build(void)
{
   memset(nedge, 0, sizeof(*nedge) * (nvert * 2));
   for ( int i = 0; i < nvert; i++ )
   for ( int j = 0; j < deg[i]; j++ ) {
      int k = G[i][j];
      if ( i > k ) continue;
      for ( int a = 0; a < nlist[i]; a++ )
      for ( int b = 0; b < nlist[k]; b++ ) {
         if ( list[i][a] == list[k][b] ) {
            Add(i, a, k, b);
         }
      }
   }
}

void
Undo(int v)
{
   if ( vis[v] == 0 ) return;
   vis[v] = 0;
   col[v] = -1;
   int vv = v * 2;
   for ( int i = 0; i < nedge[vv]; i++ )
      Undo(edge[vv][i] / 2);
}

bool
DFS(int v, int c)
{
```

```
    if ( col[v] >= 0 && col[v] != c ) return 0;
    if ( col[v] >= 0 ) return 1;
    if ( c == 1 && nlist[v] == 1 ) return 0;
    int i;
    col[v] = c;
    int vv = v * 2 + c;
    for ( i = 0; i < nedge[vv]; i++ ) {
        if ( ! DFS(edge[vv][i] / 2, edge[vv][i] % 2) ) break;
    }
    if ( i < nedge[vv] ) {
        for ( int j = 0; j < i; j++ ) Undo(edge[vv][j] / 2);
        col[v] = -1;
        return 0;
    } else {
        vis[v] = 1;
        return 1;
    }
}

void
Solve(void)
{
    int ndom = FindDom(0, 0);
    //printf("ndom=%d\n", ndom);
    int k = 1;
    for ( int i = 1; i < ndom; i++ ) k *= 3;
    domcol[0] = 0;
    for ( int s = 0; s < k; s++ ) {
        int c = s;
        for ( int i = 1; i < ndom; i++ ) {
            domcol[i] = c % 3;
            c /= 3;
        }
        int i;
        for ( i = 0; i < nvert; i++ ) {
            int r;
            if ( (r = InDom(i, ndom)) >= 0 ) {
                memset(has, 0, 3);
                has[domcol[r]] = 1;
            } else {
                memset(has, 1, 3);
                for ( int j = 0; j < deg[i]; j++ ) {
                    if ( (r = InDom(G[i][j], ndom)) >= 0 )
                        has[domcol[r]] = 0;
                }
            }
            nlist[i] = 0;
```

```
            for ( int j = 0; j < 3; j++ ) if ( has[j] )
               list[i][nlist[i]++] = j;
            if ( nlist[i] == 0 ) break;
         }
         if ( i < nvert ) continue;
         Build();
         memset(col, -1, sizeof(*col) * nvert);
         memset(vis, 0, sizeof(*vis) * nvert);
         for ( i = 0; i < nvert; i++ ) {
            if ( col[i] >= 0 ) continue;
            if ( ! DFS(i, 0) && ! DFS(i, 1) ) break;
         }
         if ( i == nvert ) {
            for ( int j = 0; j < nvert; j++ ) {
               if ( j ) printf(" ");
               printf("%d", list[j][col[j]]);
            }
            printf("\n");
            return;
         }
      }
      printf("The agents cannot be split\n");
}

int
main()
{
   while ( scanf("%d %d", &nvert, &ne), nvert ) {
      memset(deg, 0, sizeof(*deg) * nvert);
      for ( int i = 0; i < ne; i++ ) {
         int a, b;
         scanf("%d %d", &a, &b);
         G[a][deg[a]++] = b;
         G[b][deg[b]++] = a;
      }
      Solve();
   }
}
```

附录

Theoretical Computer Science Cheat Sheet

Definitions

$f(n) = O(g(n))$	iff $\exists$ positive c, n_0 such that $0 \le f(n) \le cg(n)\ \forall n \ge n_0$.
$f(n) = \Omega(g(n))$	iff $\exists$ positive c, n_0 such that $f(n) \ge cg(n) \ge 0\ \forall n \ge n_0$.
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
$f(n) = o(g(n))$	iff $\lim_{n\to\infty} f(n)/g(n) = 0$.
$\lim_{n\to\infty} a_n = a$	iff $\forall \epsilon > 0$, $\exists n_0$ such that $\lvert a_n - a\rvert < \epsilon$, $\forall n \ge n_0$.
$\sup S$	least $b \in \mathbb{R}$ such that $b \ge s$, $\forall s \in S$.
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \le s$, $\forall s \in S$.
$\liminf_{n\to\infty} a_n$	$\lim_{n\to\infty} \inf\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\limsup_{n\to\infty} a_n$	$\lim_{n\to\infty} \sup\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\binom{n}{k}$	Combinations: Size k subsets of a size n set.
$\left[{n \atop k}\right]$	Stirling numbers (1st kind): Arrangements of an n element set into k cycles.
$\left\{{n \atop k}\right\}$	Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets.
$\left\langle{n \atop k}\right\rangle$	1st order Eulerian numbers: Permutations $\pi_1\pi_2\ldots\pi_n$ on $\{1, 2, \ldots, n\}$ with k ascents.
$\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle$	2nd order Eulerian numbers.
C_n	Catalan Numbers: Binary trees with $n+1$ vertices.

Series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}.$$

In general:

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}\left((i+1)^{m+1} - i^{m+1} - (m+1)i^m\right)\right]$$

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

Geometric series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, \quad c \ne 1, \quad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \quad \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, \quad |c| < 1,$$

$$\sum_{i=0}^{n} ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \ne 1, \quad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad |c| < 1.$$

Harmonic series:

$$H_n = \sum_{i=1}^{n}\frac{1}{i}, \qquad \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n, \quad \sum_{i=1}^{n}\binom{i}{m}H_i = \binom{n+1}{m+1}\left(H_{n+1} - \frac{1}{m+1}\right).$$

1. $\binom{n}{k} = \frac{n!}{(n-k)!k!}$, **2.** $\sum_{k=0}^{n}\binom{n}{k} = 2^n$, **3.** $\binom{n}{k} = \binom{n}{n-k}$,

4. $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, **5.** $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,

6. $\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, **7.** $\sum_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n}$,

8. $\sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$, **9.** $\sum_{k=0}^{n}\binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$,

10. $\binom{n}{k} = (-1)^k\binom{k-n-1}{k}$, **11.** $\left\{{n \atop 1}\right\} = \left\{{n \atop n}\right\} = 1$,

12. $\left\{{n \atop 2}\right\} = 2^{n-1} - 1$, **13.** $\left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\}$,

14. $\left[{n \atop 1}\right] = (n-1)!$, **15.** $\left[{n \atop 2}\right] = (n-1)!H_{n-1}$, **16.** $\left[{n \atop n}\right] = 1$, **17.** $\left[{n \atop k}\right] \ge \left\{{n \atop k}\right\}$,

18. $\left[{n \atop k}\right] = (n-1)\left[{n-1 \atop k}\right] + \left[{n-1 \atop k-1}\right]$, **19.** $\left\{{n \atop n-1}\right\} = \left[{n \atop n-1}\right] = \binom{n}{2}$, **20.** $\sum_{k=0}^{n}\left[{n \atop k}\right] = n!$, **21.** $C_n = \frac{1}{n+1}\binom{2n}{n}$,

22. $\left\langle{n \atop 0}\right\rangle = \left\langle{n \atop n-1}\right\rangle = 1$, **23.** $\left\langle{n \atop k}\right\rangle = \left\langle{n \atop n-1-k}\right\rangle$, **24.** $\left\langle{n \atop k}\right\rangle = (k+1)\left\langle{n-1 \atop k}\right\rangle + (n-k)\left\langle{n-1 \atop k-1}\right\rangle$,

25. $\left\langle{0 \atop k}\right\rangle = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{otherwise} \end{cases}$ **26.** $\left\langle{n \atop 1}\right\rangle = 2^n - n - 1$, **27.** $\left\langle{n \atop 2}\right\rangle = 3^n - (n+1)2^n + \binom{n+1}{2}$,

28. $x^n = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{x+k}{n}$, **29.** $\left\langle{n \atop m}\right\rangle = \sum_{k=0}^{m}\binom{n+1}{k}(m+1-k)^n(-1)^k$, **30.** $m!\left\{{n \atop m}\right\} = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{k}{n-m}$,

31. $\left\langle{n \atop m}\right\rangle = \sum_{k=0}^{n}\left\{{n \atop k}\right\}\binom{n-k}{m}(-1)^{n-k-m}k!$, **32.** $\left\langle\!\left\langle{n \atop 0}\right\rangle\!\right\rangle = 1$, **33.** $\left\langle\!\left\langle{n \atop n}\right\rangle\!\right\rangle = 0$ for $n \ne 0$,

34. $\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = (k+1)\left\langle\!\left\langle{n-1 \atop k}\right\rangle\!\right\rangle + (2n-1-k)\left\langle\!\left\langle{n-1 \atop k-1}\right\rangle\!\right\rangle$, **35.** $\sum_{k=0}^{n}\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = \frac{(2n)^{\underline{n}}}{2^n}$,

36. $\left\{{x \atop x-n}\right\} = \sum_{k=0}^{n}\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle\binom{x+n-1-k}{2n}$, **37.** $\left\{{n+1 \atop m+1}\right\} = \sum_{k}\binom{n}{k}\left\{{k \atop m}\right\} = \sum_{k=0}^{n}\left\{{k \atop m}\right\}(m+1)^{n-k}$,

Theoretical Computer Science Cheat Sheet

Identities Cont.

38. $\begin{bmatrix} n+1 \\ m+1 \end{bmatrix} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} \binom{k}{m} = \sum_{k=0}^{n} \begin{bmatrix} k \\ m \end{bmatrix} n^{\underline{n-k}} = n! \sum_{k=0}^{n} \frac{1}{k!} \begin{bmatrix} k \\ m \end{bmatrix}$,

39. $\begin{bmatrix} x \\ x-n \end{bmatrix} = \sum_{k=0}^{n} \left\langle\!\left\langle {n \atop k} \right\rangle\!\right\rangle \binom{x+k}{2n}$,

40. $\begin{Bmatrix} n \\ m \end{Bmatrix} = \sum_k \binom{n}{k} \begin{Bmatrix} k+1 \\ m+1 \end{Bmatrix} (-1)^{n-k}$,

41. $\begin{bmatrix} n \\ m \end{bmatrix} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \binom{k}{m} (-1)^{m-k}$,

42. $\begin{Bmatrix} m+n+1 \\ m \end{Bmatrix} = \sum_{k=0}^{m} k \begin{Bmatrix} n+k \\ k \end{Bmatrix}$,

43. $\begin{bmatrix} m+n+1 \\ m \end{bmatrix} = \sum_{k=0}^{m} k(n+k) \begin{bmatrix} n+k \\ k \end{bmatrix}$,

44. $\binom{n}{m} = \sum_k \begin{Bmatrix} n+1 \\ k+1 \end{Bmatrix} \begin{bmatrix} k \\ m \end{bmatrix} (-1)^{m-k}$,

45. $(n-m)! \binom{n}{m} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \begin{Bmatrix} k \\ m \end{Bmatrix} (-1)^{m-k}$, for $n \geq m$,

46. $\begin{Bmatrix} n \\ n-m \end{Bmatrix} = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \begin{bmatrix} m+k \\ k \end{bmatrix}$,

47. $\begin{bmatrix} n \\ n-m \end{bmatrix} = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \begin{Bmatrix} m+k \\ k \end{Bmatrix}$,

48. $\begin{Bmatrix} n \\ \ell+m \end{Bmatrix} \binom{\ell+m}{\ell} = \sum_k \begin{Bmatrix} k \\ \ell \end{Bmatrix} \begin{Bmatrix} n-k \\ m \end{Bmatrix} \binom{n}{k}$,

49. $\begin{bmatrix} n \\ \ell+m \end{bmatrix} \binom{\ell+m}{\ell} = \sum_k \begin{bmatrix} k \\ \ell \end{bmatrix} \begin{bmatrix} n-k \\ m \end{bmatrix} \binom{n}{k}$.

Trees

Every tree with n vertices has $n-1$ edges.

Kraft inequality: If the depths of the leaves of a binary tree are $d_1, \ldots, d_n$:

$$\sum_{i=1}^{n} 2^{-d_i} \leq 1,$$

and equality holds only if every internal node has 2 sons.

Recurrences

Master method:
$T(n) = aT(n/b) + f(n), \quad a \geq 1, b > 1$

If $\exists \epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$ then

$$T(n) = \Theta(n^{\log_b a}).$$

If $f(n) = \Theta(n^{\log_b a})$ then

$$T(n) = \Theta(n^{\log_b a} \log_2 n).$$

If $\exists \epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and $\exists c < 1$ such that $af(n/b) \leq cf(n)$ for large n, then

$$T(n) = \Theta(f(n)).$$

Substitution (example): Consider the following recurrence

$$T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.$$

Note that T_i is always a power of two. Let $t_i = \log_2 T_i$. Then we have

$$t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.$$

Let $u_i = t_i / 2^i$. Dividing both sides of the previous equation by 2^{i+1} we get

$$\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.$$

Substituting we find

$$u_{i+1} = \tfrac{1}{2} + u_i, \qquad u_1 = \tfrac{1}{2},$$

which is simply $u_i = i/2$. So we find that T_i has the closed form $T_i = 2^{i2^{i-1}}$.

Summing factors (example): Consider the following recurrence

$$T(n) = 3T(n/2) + n, \quad T(1) = 1.$$

Rewrite so that all terms involving T are on the left side

$$T(n) - 3T(n/2) = n.$$

Now expand the recurrence, and choose a factor which makes the left side "telescope"

$$\begin{aligned} &1\big(T(n) - 3T(n/2) = n\big) \\ &3\big(T(n/2) - 3T(n/4) = n/2\big) \\ &\quad \vdots \quad\quad \vdots \quad\quad \vdots \\ &3^{\log_2 n - 1}\big(T(2) - 3T(1) = 2\big) \end{aligned}$$

Let $m = \log_2 n$. Summing the left side we get $T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k$ where $k = \log_2 3 \approx 1.58496$. Summing the right side we get

$$\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n \sum_{i=0}^{m-1} \left(\tfrac{3}{2}\right)^i.$$

Let $c = \frac{3}{2}$. Then we have

$$\begin{aligned} n \sum_{i=0}^{m-1} c^i &= n \left(\frac{c^m - 1}{c - 1} \right) \\ &= 2n(c^{\log_2 n} - 1) \\ &= 2n(c^{(k-1)\log_c n} - 1) \\ &= 2n^k - 2n, \end{aligned}$$

and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider

$$T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$$

Note that

$$T_{i+1} = 1 + \sum_{j=0}^{i} T_j.$$

Subtracting we find

$$\begin{aligned} T_{i+1} - T_i &= 1 + \sum_{j=0}^{i} T_j - 1 - \sum_{j=0}^{i-1} T_j \\ &= T_i. \end{aligned}$$

And so $T_{i+1} = 2T_i = 2^{i+1}$.

Generating functions:

1. Multiply both sides of the equation by x^i.
2. Sum both sides over all i for which the equation is valid.
3. Choose a generating function $G(x)$. Usually $G(x) = \sum_{i=0}^{\infty} x^i g_i$.
3. Rewrite the equation in terms of the generating function $G(x)$.
4. Solve for $G(x)$.
5. The coefficient of x^i in $G(x)$ is g_i.

Example:

$$g_{i+1} = 2g_i + 1, \quad g_0 = 0.$$

Multiply and sum:

$$\sum_{i \geq 0} g_{i+1} x^i = \sum_{i \geq 0} 2g_i x^i + \sum_{i \geq 0} x^i.$$

We choose $G(x) = \sum_{i \geq 0} x^i g_i$. Rewrite in terms of $G(x)$:

$$\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \geq 0} x^i.$$

Simplify:

$$\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$$

Solve for $G(x)$:

$$G(x) = \frac{x}{(1-x)(1-2x)}.$$

Expand this using partial fractions:

$$\begin{aligned} G(x) &= x \left(\frac{2}{1-2x} - \frac{1}{1-x} \right) \\ &= x \left(2 \sum_{i \geq 0} 2^i x^i - \sum_{i \geq 0} x^i \right) \\ &= \sum_{i \geq 0} (2^{i+1} - 1) x^{i+1}. \end{aligned}$$

So $g_i = 2^i - 1$.

Theoretical Computer Science Cheat Sheet

$\pi \approx 3.14159$, $e \approx 2.71828$, $\gamma \approx 0.57721$, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803$, $\hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -.61803$

i	2^i	p_i
1	2	2
2	4	3
3	8	5
4	16	7
5	32	11
6	64	13
7	128	17
8	256	19
9	512	23
10	1,024	29
11	2,048	31
12	4,096	37
13	8,192	41
14	16,384	43
15	32,768	47
16	65,536	53
17	131,072	59
18	262,144	61
19	524,288	67
20	1,048,576	71
21	2,097,152	73
22	4,194,304	79
23	8,388,608	83
24	16,777,216	89
25	33,554,432	97
26	67,108,864	101
27	134,217,728	103
28	268,435,456	107
29	536,870,912	109
30	1,073,741,824	113
31	2,147,483,648	127
32	4,294,967,296	131

Pascal's Triangle

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1

General

Bernoulli Numbers ($B_i = 0$, odd $i \neq 1$):

$B_0 = 1$, $B_1 = -\frac{1}{2}$, $B_2 = \frac{1}{6}$, $B_4 = -\frac{1}{30}$, $B_6 = \frac{1}{42}$, $B_8 = -\frac{1}{30}$, $B_{10} = \frac{5}{66}$.

Change of base, quadratic formula:

$$\log_b x = \frac{\log_a x}{\log_a b}, \qquad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Euler's number e:

$$e = 1 + \tfrac{1}{2} + \tfrac{1}{6} + \tfrac{1}{24} + \tfrac{1}{120} + \cdots$$

$$\lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n = e^x.$$

$$\left(1 + \tfrac{1}{n}\right)^n < e < \left(1 + \tfrac{1}{n}\right)^{n+1}.$$

$$\left(1 + \tfrac{1}{n}\right)^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$$

Harmonic numbers:

$1, \frac{3}{2}, \frac{11}{6}, \frac{25}{12}, \frac{137}{60}, \frac{49}{20}, \frac{363}{140}, \frac{761}{280}, \frac{7129}{2520}, \ldots$

$$\ln n < H_n < \ln n + 1,$$

$$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right).$$

Factorial, Stirling's approximation:

1, 2, 6, 24, 120, 720, 5040, 40320, 362880, ...

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$$

Ackermann's function and inverse:

$$a(i,j) = \begin{cases} 2^j & i = 1 \\ a(i-1,2) & j = 1 \\ a(i-1, a(i,j-1)) & i,j \geq 2 \end{cases}$$

$$\alpha(i) = \min\{j \mid a(j,j) \geq i\}.$$

Binomial distribution:

$$\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \qquad q = 1 - p,$$

$$\mathrm{E}[X] = \sum_{k=1}^{n} k \binom{n}{k} p^k q^{n-k} = np.$$

Poisson distribution:

$$\Pr[X = k] = \frac{e^{-\lambda}\lambda^k}{k!}, \quad \mathrm{E}[X] = \lambda.$$

Normal (Gaussian) distribution:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}, \quad \mathrm{E}[X] = \mu.$$

The "coupon collector": We are given a random coupon each day, and there are n different types of coupons. The distribution of coupons is uniform. The expected number of days to pass before we to collect all n types is

$$nH_n.$$

Probability

Continuous distributions: If

$$\Pr[a < X < b] = \int_a^b p(x)\,dx,$$

then p is the probability density function of X. If

$$\Pr[X < a] = P(a),$$

then P is the distribution function of X. If P and p both exist then

$$P(a) = \int_{-\infty}^{a} p(x)\,dx.$$

Expectation: If X is discrete

$$\mathrm{E}[g(X)] = \sum_x g(x) \Pr[X = x].$$

If X continuous then

$$\mathrm{E}[g(X)] = \int_{-\infty}^{\infty} g(x)p(x)\,dx = \int_{-\infty}^{\infty} g(x)\,dP(x).$$

Variance, standard deviation:

$$\mathrm{VAR}[X] = \mathrm{E}[X^2] - \mathrm{E}[X]^2,$$

$$\sigma = \sqrt{\mathrm{VAR}[X]}.$$

For events A and B:

$$\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]$$

$$\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$$

iff A and B are independent.

$$\Pr[A|B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$$

For random variables X and Y:

$$\mathrm{E}[X \cdot Y] = \mathrm{E}[X] \cdot \mathrm{E}[Y],$$

if X and Y are independent.

$$\mathrm{E}[X + Y] = \mathrm{E}[X] + \mathrm{E}[Y],$$

$$\mathrm{E}[cX] = c\,\mathrm{E}[X].$$

Bayes' theorem:

$$\Pr[A_i|B] = \frac{\Pr[B|A_i]\Pr[A_i]}{\sum_{j=1}^{n} \Pr[A_j]\Pr[B|A_j]}.$$

Inclusion-exclusion:

$$\Pr\left[\bigvee_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \Pr[X_i] + \sum_{k=2}^{n} (-1)^{k+1} \sum_{i_i < \cdots < i_k} \Pr\left[\bigwedge_{j=1}^{k} X_{i_j}\right].$$

Moment inequalities:

$$\Pr\left[|X| \geq \lambda\,\mathrm{E}[X]\right] \leq \frac{1}{\lambda},$$

$$\Pr\left[\left|X - \mathrm{E}[X]\right| \geq \lambda \cdot \sigma\right] \leq \frac{1}{\lambda^2}.$$

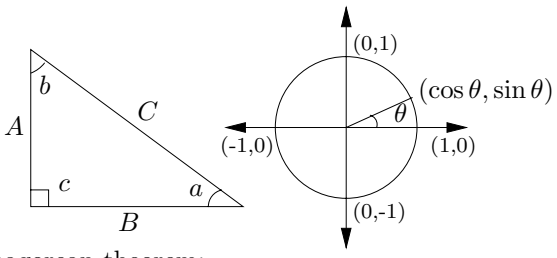
Geometric distribution:

$$\Pr[X = k] = pq^{k-1}, \qquad q = 1 - p,$$

$$\mathrm{E}[X] = \sum_{k=1}^{\infty} kpq^{k-1} = \frac{1}{p}.$$

Theoretical Computer Science Cheat Sheet

Trigonometry



Pythagorean theorem:

$$C^2 = A^2 + B^2.$$

Definitions:

$$\sin a = A/C, \quad \cos a = B/C,$$
$$\csc a = C/A, \quad \sec a = C/B,$$
$$\tan a = \frac{\sin a}{\cos a} = \frac{A}{B}, \quad \cot a = \frac{\cos a}{\sin a} = \frac{B}{A}.$$

Area, radius of inscribed circle:

$$\tfrac{1}{2}AB, \quad \frac{AB}{A+B+C}.$$

Identities:

$$\sin x = \frac{1}{\csc x}, \qquad \cos x = \frac{1}{\sec x},$$
$$\tan x = \frac{1}{\cot x}, \qquad \sin^2 x + \cos^2 x = 1,$$
$$1 + \tan^2 x = \sec^2 x, \qquad 1 + \cot^2 x = \csc^2 x,$$
$$\sin x = \cos\left(\tfrac{\pi}{2} - x\right), \qquad \sin x = \sin(\pi - x),$$
$$\cos x = -\cos(\pi - x), \qquad \tan x = \cot\left(\tfrac{\pi}{2} - x\right),$$
$$\cot x = -\cot(\pi - x), \qquad \csc x = \cot\tfrac{x}{2} - \cot x,$$
$$\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y,$$
$$\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y,$$
$$\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$$
$$\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$$
$$\sin 2x = 2\sin x \cos x, \qquad \sin 2x = \frac{2\tan x}{1 + \tan^2 x},$$
$$\cos 2x = \cos^2 x - \sin^2 x, \qquad \cos 2x = 2\cos^2 x - 1,$$
$$\cos 2x = 1 - 2\sin^2 x, \qquad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$$
$$\tan 2x = \frac{2\tan x}{1 - \tan^2 x}, \qquad \cot 2x = \frac{\cot^2 x - 1}{2\cot x},$$
$$\sin(x + y)\sin(x - y) = \sin^2 x - \sin^2 y,$$
$$\cos(x + y)\cos(x - y) = \cos^2 x - \sin^2 y.$$

Euler's equation:

$$e^{ix} = \cos x + i\sin x, \qquad e^{i\pi} = -1.$$

v2.02 ©1994 by Steve Seiden
sseiden@acm.org
http://www.csc.lsu.edu/~seiden

Matrices

Multiplication:

$$C = A \cdot B, \quad c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}.$$

Determinants: $\det A \neq 0$ iff A is non-singular.

$$\det A \cdot B = \det A \cdot \det B,$$
$$\det A = \sum_{\pi} \prod_{i=1}^{n} \operatorname{sign}(\pi) a_{i,\pi(i)}.$$

2×2 and 3×3 determinant:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = g\begin{vmatrix} b & c \\ e & f \end{vmatrix} - h\begin{vmatrix} a & c \\ d & f \end{vmatrix} + i\begin{vmatrix} a & b \\ d & e \end{vmatrix}$$
$$= aei + bfg + cdh - ceg - fha - ibd.$$

Permanents:

$$\operatorname{perm} A = \sum_{\pi} \prod_{i=1}^{n} a_{i,\pi(i)}.$$

Hyperbolic Functions

Definitions:

$$\sinh x = \frac{e^x - e^{-x}}{2}, \qquad \cosh x = \frac{e^x + e^{-x}}{2},$$
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad \operatorname{csch} x = \frac{1}{\sinh x},$$
$$\operatorname{sech} x = \frac{1}{\cosh x}, \qquad \coth x = \frac{1}{\tanh x}.$$

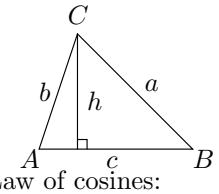
Identities:

$$\cosh^2 x - \sinh^2 x = 1, \qquad \tanh^2 x + \operatorname{sech}^2 x = 1,$$
$$\coth^2 x - \operatorname{csch}^2 x = 1, \qquad \sinh(-x) = -\sinh x,$$
$$\cosh(-x) = \cosh x, \qquad \tanh(-x) = -\tanh x,$$
$$\sinh(x + y) = \sinh x \cosh y + \cosh x \sinh y,$$
$$\cosh(x + y) = \cosh x \cosh y + \sinh x \sinh y,$$
$$\sinh 2x = 2\sinh x \cosh x,$$
$$\cosh 2x = \cosh^2 x + \sinh^2 x,$$
$$\cosh x + \sinh x = e^x, \qquad \cosh x - \sinh x = e^{-x},$$
$$(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$$
$$2\sinh^2 \tfrac{x}{2} = \cosh x - 1, \qquad 2\cosh^2 \tfrac{x}{2} = \cosh x + 1.$$

θ	$\sin\theta$	$\cos\theta$	$\tan\theta$
0	0	1	0
$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{3}}{3}$
$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	1
$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	$\sqrt{3}$
$\frac{\pi}{2}$	1	0	∞

. . . in mathematics you don't understand things, you just get used to them.
– J. von Neumann

More Trig.



Law of cosines:

$$c^2 = a^2 + b^2 - 2ab\cos C.$$

Area:

$$A = \tfrac{1}{2}hc,$$
$$= \tfrac{1}{2}ab\sin C,$$
$$= \frac{c^2 \sin A \sin B}{2\sin C}.$$

Heron's formula:

$$A = \sqrt{s \cdot s_a \cdot s_b \cdot s_c},$$
$$s = \tfrac{1}{2}(a + b + c),$$
$$s_a = s - a,$$
$$s_b = s - b,$$
$$s_c = s - c.$$

More identities:

$$\sin\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{2}},$$
$$\cos\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{2}},$$
$$\tan\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}},$$
$$= \frac{1 - \cos x}{\sin x},$$
$$= \frac{\sin x}{1 + \cos x},$$
$$\cot\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{1 - \cos x}},$$
$$= \frac{1 + \cos x}{\sin x},$$
$$= \frac{\sin x}{1 - \cos x},$$
$$\sin x = \frac{e^{ix} - e^{-ix}}{2i},$$
$$\cos x = \frac{e^{ix} + e^{-ix}}{2},$$
$$\tan x = -i\frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}},$$
$$= -i\frac{e^{2ix} - 1}{e^{2ix} + 1},$$
$$\sin x = \frac{\sinh ix}{i},$$
$$\cos x = \cosh ix,$$
$$\tan x = \frac{\tanh ix}{i}.$$

Theoretical Computer Science Cheat Sheet

Number Theory

The Chinese remainder theorem: There exists a number C such that:

$$\begin{aligned} C &\equiv r_1 \bmod m_1 \\ &\vdots \quad \vdots \quad \vdots \\ C &\equiv r_n \bmod m_n \end{aligned}$$

if m_i and m_j are relatively prime for $i \neq j$.

Euler's function: $\phi(x)$ is the number of positive integers less than x relatively prime to x. If $\prod_{i=1}^{n} p_i^{e_i}$ is the prime factorization of x then

$$\phi(x) = \prod_{i=1}^{n} p_i^{e_i-1}(p_i - 1).$$

Euler's theorem: If a and b are relatively prime then

$$1 \equiv a^{\phi(b)} \bmod b.$$

Fermat's theorem:

$$1 \equiv a^{p-1} \bmod p.$$

The Euclidean algorithm: if $a > b$ are integers then

$$\gcd(a, b) = \gcd(a \bmod b, b).$$

If $\prod_{i=1}^{n} p_i^{e_i}$ is the prime factorization of x then

$$S(x) = \sum_{d|x} d = \prod_{i=1}^{n} \frac{p_i^{e_i+1} - 1}{p_i - 1}.$$

Perfect Numbers: x is an even perfect number iff $x = 2^{n-1}(2^n - 1)$ and $2^n - 1$ is prime.

Wilson's theorem: n is a prime iff

$$(n-1)! \equiv -1 \bmod n.$$

Möbius inversion:

$$\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of } r \text{ distinct primes.} \end{cases}$$

If

$$G(a) = \sum_{d|a} F(d),$$

then

$$F(a) = \sum_{d|a} \mu(d) G\Big(\frac{a}{d}\Big).$$

Prime numbers:

$$p_n = n \ln n + n \ln \ln n - n + n\frac{\ln \ln n}{\ln n} + O\Big(\frac{n}{\ln n}\Big),$$

$$\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3} + O\Big(\frac{n}{(\ln n)^4}\Big).$$

Graph Theory

Definitions:

Loop	An edge connecting a vertex to itself.
Directed	Each edge has a direction.
Simple	Graph with no loops or multi-edges.
Walk	A sequence $v_0 e_1 v_1 \ldots e_\ell v_\ell$.
Trail	A walk with distinct edges.
Path	A trail with distinct vertices.
Connected	A graph where there exists a path between any two vertices.
Component	A maximal connected subgraph.
Tree	A connected acyclic graph.
Free tree	A tree with no root.
DAG	Directed acyclic graph.
Eulerian	Graph with a trail visiting each edge exactly once.
Hamiltonian	Graph with a cycle visiting each vertex exactly once.
Cut	A set of edges whose removal increases the number of components.
Cut-set	A minimal cut.
Cut edge	A size 1 cut.
k-Connected	A graph connected with the removal of any $k-1$ vertices.
k-Tough	$\forall S \subseteq V, S \neq \emptyset$ we have $k \cdot c(G-S) \leq \lvert S \rvert$.
k-Regular	A graph where all vertices have degree k.
k-Factor	A k-regular spanning subgraph.
Matching	A set of edges, no two of which are adjacent.
Clique	A set of vertices, all of which are adjacent.
Ind. set	A set of vertices, none of which are adjacent.
Vertex cover	A set of vertices which cover all edges.
Planar graph	A graph which can be embeded in the plane.
Plane graph	An embedding of a planar graph.

$$\sum_{v \in V} \deg(v) = 2m.$$

If G is planar then $n - m + f = 2$, so

$$f \leq 2n - 4, \quad m \leq 3n - 6.$$

Any planar graph has a vertex with degree ≤ 5.

Notation:

$E(G)$	Edge set
$V(G)$	Vertex set
$c(G)$	Number of components
$G[S]$	Induced subgraph
$\deg(v)$	Degree of v
$\Delta(G)$	Maximum degree
$\delta(G)$	Minimum degree
$\chi(G)$	Chromatic number
$\chi_E(G)$	Edge chromatic number
G^c	Complement graph
K_n	Complete graph
K_{n_1,n_2}	Complete bipartite graph
$\mathrm{r}(k, \ell)$	Ramsey number

Geometry

Projective coordinates: triples (x, y, z), not all x, y and z zero.

$$(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$$

Cartesian	Projective
(x, y)	$(x, y, 1)$
$y = mx + b$	$(m, -1, b)$
$x = c$	$(1, 0, -c)$

Distance formula, L_p and L_∞ metric:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$

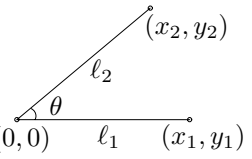
$$\left[|x_1 - x_0|^p + |y_1 - y_0|^p\right]^{1/p},$$

$$\lim_{p \to \infty} \left[|x_1 - x_0|^p + |y_1 - y_0|^p\right]^{1/p}.$$

Area of triangle (x_0, y_0), (x_1, y_1) and (x_2, y_2):

$$\tfrac{1}{2} \operatorname{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$$

Angle formed by three points:



$$\cos\theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{\ell_1 \ell_2}.$$

Line through two points (x_0, y_0) and (x_1, y_1):

$$\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$$

Area of circle, volume of sphere:

$$A = \pi r^2, \qquad V = \tfrac{4}{3}\pi r^3.$$

If I have seen farther than others, it is because I have stood on the shoulders of giants.
– Issac Newton

Theoretical Computer Science Cheat Sheet

π

Wallis' identity:

$$\pi = 2 \cdot \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdots}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdots}$$

Brouncker's continued fraction expansion:

$$\frac{\pi}{4} = 1 + \cfrac{1^2}{2 + \cfrac{3^2}{2 + \cfrac{5^2}{2 + \cfrac{7^2}{2 + \cdots}}}}$$

Gregrory's series:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

Newton's series:

$$\frac{\pi}{6} = \frac{1}{2} + \frac{1}{2 \cdot 3 \cdot 2^3} + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5 \cdot 2^5} + \cdots$$

Sharp's series:

$$\frac{\pi}{6} = \frac{1}{\sqrt{3}}\left(1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \cdots\right)$$

Euler's series:

$$\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \cdots$$
$$\frac{\pi^2}{8} = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \frac{1}{9^2} + \cdots$$
$$\frac{\pi^2}{12} = \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \cdots$$

Partial Fractions

Let $N(x)$ and $D(x)$ be polynomial functions of x. We can break down $N(x)/D(x)$ using partial fraction expansion. First, if the degree of N is greater than or equal to the degree of D, divide N by D, obtaining

$$\frac{N(x)}{D(x)} = Q(x) + \frac{N'(x)}{D(x)},$$

where the degree of N' is less than that of D. Second, factor $D(x)$. Use the following rules: For a non-repeated factor:

$$\frac{N(x)}{(x-a)D(x)} = \frac{A}{x-a} + \frac{N'(x)}{D(x)},$$

where

$$A = \left[\frac{N(x)}{D(x)}\right]_{x=a}.$$

For a repeated factor:

$$\frac{N(x)}{(x-a)^m D(x)} = \sum_{k=0}^{m-1} \frac{A_k}{(x-a)^{m-k}} + \frac{N'(x)}{D(x)},$$

where

$$A_k = \frac{1}{k!}\left[\frac{d^k}{dx^k}\left(\frac{N(x)}{D(x)}\right)\right]_{x=a}.$$

The reasonable man adapts himself to the world; the unreasonable persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable. – George Bernard Shaw

Calculus

Derivatives:

1. $\frac{d(cu)}{dx} = c\frac{du}{dx}$, **2.** $\frac{d(u+v)}{dx} = \frac{du}{dx} + \frac{dv}{dx}$, **3.** $\frac{d(uv)}{dx} = u\frac{dv}{dx} + v\frac{du}{dx}$,

4. $\frac{d(u^n)}{dx} = nu^{n-1}\frac{du}{dx}$, **5.** $\frac{d(u/v)}{dx} = \frac{v\left(\frac{du}{dx}\right) - u\left(\frac{dv}{dx}\right)}{v^2}$, **6.** $\frac{d(e^{cu})}{dx} = ce^{cu}\frac{du}{dx}$,

7. $\frac{d(c^u)}{dx} = (\ln c)c^u\frac{du}{dx}$, **8.** $\frac{d(\ln u)}{dx} = \frac{1}{u}\frac{du}{dx}$,

9. $\frac{d(\sin u)}{dx} = \cos u\frac{du}{dx}$, **10.** $\frac{d(\cos u)}{dx} = -\sin u\frac{du}{dx}$,

11. $\frac{d(\tan u)}{dx} = \sec^2 u\frac{du}{dx}$, **12.** $\frac{d(\cot u)}{dx} = \csc^2 u\frac{du}{dx}$,

13. $\frac{d(\sec u)}{dx} = \tan u\,\sec u\frac{du}{dx}$, **14.** $\frac{d(\csc u)}{dx} = -\cot u\,\csc u\frac{du}{dx}$,

15. $\frac{d(\arcsin u)}{dx} = \frac{1}{\sqrt{1-u^2}}\frac{du}{dx}$, **16.** $\frac{d(\arccos u)}{dx} = \frac{-1}{\sqrt{1-u^2}}\frac{du}{dx}$,

17. $\frac{d(\arctan u)}{dx} = \frac{1}{1+u^2}\frac{du}{dx}$, **18.** $\frac{d(\operatorname{arccot} u)}{dx} = \frac{-1}{1+u^2}\frac{du}{dx}$,

19. $\frac{d(\operatorname{arcsec} u)}{dx} = \frac{1}{u\sqrt{1-u^2}}\frac{du}{dx}$, **20.** $\frac{d(\operatorname{arccsc} u)}{dx} = \frac{-1}{u\sqrt{1-u^2}}\frac{du}{dx}$,

21. $\frac{d(\sinh u)}{dx} = \cosh u\frac{du}{dx}$, **22.** $\frac{d(\cosh u)}{dx} = \sinh u\frac{du}{dx}$,

23. $\frac{d(\tanh u)}{dx} = \operatorname{sech}^2 u\frac{du}{dx}$, **24.** $\frac{d(\coth u)}{dx} = -\operatorname{csch}^2 u\frac{du}{dx}$,

25. $\frac{d(\operatorname{sech} u)}{dx} = -\operatorname{sech} u\,\tanh u\frac{du}{dx}$, **26.** $\frac{d(\operatorname{csch} u)}{dx} = -\operatorname{csch} u\,\coth u\frac{du}{dx}$,

27. $\frac{d(\operatorname{arcsinh} u)}{dx} = \frac{1}{\sqrt{1+u^2}}\frac{du}{dx}$, **28.** $\frac{d(\operatorname{arccosh} u)}{dx} = \frac{1}{\sqrt{u^2-1}}\frac{du}{dx}$,

29. $\frac{d(\operatorname{arctanh} u)}{dx} = \frac{1}{1-u^2}\frac{du}{dx}$, **30.** $\frac{d(\operatorname{arccoth} u)}{dx} = \frac{1}{u^2-1}\frac{du}{dx}$,

31. $\frac{d(\operatorname{arcsech} u)}{dx} = \frac{-1}{u\sqrt{1-u^2}}\frac{du}{dx}$, **32.** $\frac{d(\operatorname{arccsch} u)}{dx} = \frac{-1}{|u|\sqrt{1+u^2}}\frac{du}{dx}$.

Integrals:

1. $\int cu\,dx = c\int u\,dx$, **2.** $\int (u+v)\,dx = \int u\,dx + \int v\,dx$,

3. $\int x^n\,dx = \frac{1}{n+1}x^{n+1}, \quad n \neq -1$, **4.** $\int \frac{1}{x}dx = \ln x$, **5.** $\int e^x\,dx = e^x$,

6. $\int \frac{dx}{1+x^2} = \arctan x$, **7.** $\int u\frac{dv}{dx}dx = uv - \int v\frac{du}{dx}dx$,

8. $\int \sin x\,dx = -\cos x$, **9.** $\int \cos x\,dx = \sin x$,

10. $\int \tan x\,dx = -\ln|\cos x|$, **11.** $\int \cot x\,dx = \ln|\cos x|$,

12. $\int \sec x\,dx = \ln|\sec x + \tan x|$, **13.** $\int \csc x\,dx = \ln|\csc x + \cot x|$,

14. $\int \arcsin\frac{x}{a}dx = \arcsin\frac{x}{a} + \sqrt{a^2-x^2}, \quad a > 0$,

Theoretical Computer Science Cheat Sheet

Calculus Cont.

15. $\int \arccos\frac{x}{a}dx = \arccos\frac{x}{a} - \sqrt{a^2-x^2}, \quad a>0,$

16. $\int \arctan\frac{x}{a}dx = x\arctan\frac{x}{a} - \frac{a}{2}\ln(a^2+x^2), \quad a>0,$

17. $\int \sin^2(ax)dx = \frac{1}{2a}\big(ax - \sin(ax)\cos(ax)\big),$

18. $\int \cos^2(ax)dx = \frac{1}{2a}\big(ax + \sin(ax)\cos(ax)\big),$

19. $\int \sec^2 x\,dx = \tan x,$

20. $\int \csc^2 x\,dx = -\cot x,$

21. $\int \sin^n x\,dx = -\frac{\sin^{n-1}x\cos x}{n} + \frac{n-1}{n}\int \sin^{n-2}x\,dx,$

22. $\int \cos^n x\,dx = \frac{\cos^{n-1}x\sin x}{n} + \frac{n-1}{n}\int \cos^{n-2}x\,dx,$

23. $\int \tan^n x\,dx = \frac{\tan^{n-1}x}{n-1} - \int \tan^{n-2}x\,dx, \quad n\neq 1,$

24. $\int \cot^n x\,dx = -\frac{\cot^{n-1}x}{n-1} - \int \cot^{n-2}x\,dx, \quad n\neq 1,$

25. $\int \sec^n x\,dx = \frac{\tan x\sec^{n-1}x}{n-1} + \frac{n-2}{n-1}\int \sec^{n-2}x\,dx, \quad n\neq 1,$

26. $\int \csc^n x\,dx = -\frac{\cot x\csc^{n-1}x}{n-1} + \frac{n-2}{n-1}\int \csc^{n-2}x\,dx, \quad n\neq 1,$

27. $\int \sinh x\,dx = \cosh x,$

28. $\int \cosh x\,dx = \sinh x,$

29. $\int \tanh x\,dx = \ln|\cosh x|,$

30. $\int \coth x\,dx = \ln|\sinh x|,$

31. $\int \operatorname{sech} x\,dx = \arctan\sinh x,$

32. $\int \operatorname{csch} x\,dx = \ln\left|\tanh\frac{x}{2}\right|,$

33. $\int \sinh^2 x\,dx = \frac{1}{4}\sinh(2x) - \frac{1}{2}x,$

34. $\int \cosh^2 x\,dx = \frac{1}{4}\sinh(2x) + \frac{1}{2}x,$

35. $\int \operatorname{sech}^2 x\,dx = \tanh x,$

36. $\int \operatorname{arcsinh}\frac{x}{a}dx = x\operatorname{arcsinh}\frac{x}{a} - \sqrt{x^2+a^2}, \quad a>0,$

37. $\int \operatorname{arctanh}\frac{x}{a}dx = x\operatorname{arctanh}\frac{x}{a} + \frac{a}{2}\ln|a^2-x^2|,$

38. $\int \operatorname{arccosh}\frac{x}{a}dx = \begin{cases} x\operatorname{arccosh}\dfrac{x}{a} - \sqrt{x^2+a^2}, & \text{if } \operatorname{arccosh}\frac{x}{a} > 0 \text{ and } a>0, \\ x\operatorname{arccosh}\dfrac{x}{a} + \sqrt{x^2+a^2}, & \text{if } \operatorname{arccosh}\frac{x}{a} < 0 \text{ and } a>0, \end{cases}$

39. $\int \frac{dx}{\sqrt{a^2+x^2}} = \ln\left(x+\sqrt{a^2+x^2}\right), \quad a>0,$

40. $\int \frac{dx}{a^2+x^2} = \frac{1}{a}\arctan\frac{x}{a}, \quad a>0,$

41. $\int \sqrt{a^2-x^2}\,dx = \frac{x}{2}\sqrt{a^2-x^2} + \frac{a^2}{2}\arcsin\frac{x}{a}, \quad a>0,$

42. $\int (a^2-x^2)^{3/2}dx = \frac{x}{8}(5a^2-2x^2)\sqrt{a^2-x^2} + \frac{3a^4}{8}\arcsin\frac{x}{a}, \quad a>0,$

43. $\int \frac{dx}{\sqrt{a^2-x^2}} = \arcsin\frac{x}{a}, \quad a>0,$

44. $\int \frac{dx}{a^2-x^2} = \frac{1}{2a}\ln\left|\frac{a+x}{a-x}\right|,$

45. $\int \frac{dx}{(a^2-x^2)^{3/2}} = \frac{x}{a^2\sqrt{a^2-x^2}},$

46. $\int \sqrt{a^2\pm x^2}\,dx = \frac{x}{2}\sqrt{a^2\pm x^2} \pm \frac{a^2}{2}\ln\left|x+\sqrt{a^2\pm x^2}\right|,$

47. $\int \frac{dx}{\sqrt{x^2-a^2}} = \ln\left|x+\sqrt{x^2-a^2}\right|, \quad a>0,$

48. $\int \frac{dx}{ax^2+bx} = \frac{1}{a}\ln\left|\frac{x}{a+bx}\right|,$

49. $\int x\sqrt{a+bx}\,dx = \frac{2(3bx-2a)(a+bx)^{3/2}}{15b^2},$

50. $\int \frac{\sqrt{a+bx}}{x}\,dx = 2\sqrt{a+bx} + a\int \frac{1}{x\sqrt{a+bx}}dx,$

51. $\int \frac{x}{\sqrt{a+bx}}\,dx = \frac{1}{\sqrt{2}}\ln\left|\frac{\sqrt{a+bx}-\sqrt{a}}{\sqrt{a+bx}+\sqrt{a}}\right|, \quad a>0,$

52. $\int \frac{\sqrt{a^2-x^2}}{x}\,dx = \sqrt{a^2-x^2} - a\ln\left|\frac{a+\sqrt{a^2-x^2}}{x}\right|,$

53. $\int x\sqrt{a^2-x^2}\,dx = -\frac{1}{3}(a^2-x^2)^{3/2},$

54. $\int x^2\sqrt{a^2-x^2}\,dx = \frac{x}{8}(2x^2-a^2)\sqrt{a^2-x^2} + \frac{a^4}{8}\arcsin\frac{x}{a}, \quad a>0,$

55. $\int \frac{dx}{\sqrt{a^2-x^2}} = -\frac{1}{a}\ln\left|\frac{a+\sqrt{a^2-x^2}}{x}\right|,$

56. $\int \frac{x\,dx}{\sqrt{a^2-x^2}} = -\sqrt{a^2-x^2},$

57. $\int \frac{x^2\,dx}{\sqrt{a^2-x^2}} = -\frac{x}{2}\sqrt{a^2-x^2} + \frac{a^2}{2}\arcsin\frac{x}{a}, \quad a>0,$

58. $\int \frac{\sqrt{a^2+x^2}}{x}\,dx = \sqrt{a^2+x^2} - a\ln\left|\frac{a+\sqrt{a^2+x^2}}{x}\right|,$

59. $\int \frac{\sqrt{x^2-a^2}}{x}\,dx = \sqrt{x^2-a^2} - a\arccos\frac{a}{|x|}, \quad a>0,$

60. $\int x\sqrt{x^2\pm a^2}\,dx = \frac{1}{3}(x^2\pm a^2)^{3/2},$

61. $\int \frac{dx}{x\sqrt{x^2+a^2}} = \frac{1}{a}\ln\left|\frac{x}{a+\sqrt{a^2+x^2}}\right|,$

Theoretical Computer Science Cheat Sheet

Calculus Cont.

62. $\displaystyle\int \frac{dx}{x\sqrt{x^2-a^2}} = \tfrac{1}{a}\arccos\tfrac{a}{|x|}, \quad a>0,$

63. $\displaystyle\int \frac{dx}{x^2\sqrt{x^2\pm a^2}} = \mp\frac{\sqrt{x^2\pm a^2}}{a^2x},$

64. $\displaystyle\int \frac{x\,dx}{\sqrt{x^2\pm a^2}} = \sqrt{x^2\pm a^2},$

65. $\displaystyle\int \frac{\sqrt{x^2\pm a^2}}{x^4}\,dx = \mp\frac{(x^2+a^2)^{3/2}}{3a^2x^3},$

66. $$\int \frac{dx}{ax^2+bx+c} = \begin{cases} \dfrac{1}{\sqrt{b^2-4ac}}\ln\left|\dfrac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}}\right|, & \text{if } b^2>4ac, \\ \dfrac{2}{\sqrt{4ac-b^2}}\arctan\dfrac{2ax+b}{\sqrt{4ac-b^2}}, & \text{if } b^2<4ac, \end{cases}$$

67. $$\int \frac{dx}{\sqrt{ax^2+bx+c}} = \begin{cases} \dfrac{1}{\sqrt{a}}\ln\left|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}\right|, & \text{if } a>0, \\ \dfrac{1}{\sqrt{-a}}\arcsin\dfrac{-2ax-b}{\sqrt{b^2-4ac}}, & \text{if } a<0, \end{cases}$$

68. $\displaystyle\int \sqrt{ax^2+bx+c}\,dx = \frac{2ax+b}{4a}\sqrt{ax^2+bx+c} + \frac{4ax-b^2}{8a}\int\frac{dx}{\sqrt{ax^2+bx+c}},$

69. $\displaystyle\int \frac{x\,dx}{\sqrt{ax^2+bx+c}} = \frac{\sqrt{ax^2+bx+c}}{a} - \frac{b}{2a}\int\frac{dx}{\sqrt{ax^2+bx+c}},$

70. $$\int \frac{dx}{x\sqrt{ax^2+bx+c}} = \begin{cases} \dfrac{-1}{\sqrt{c}}\ln\left|\dfrac{2\sqrt{c}\sqrt{ax^2+bx+c}+bx+2c}{x}\right|, & \text{if } c>0, \\ \dfrac{1}{\sqrt{-c}}\arcsin\dfrac{bx+2c}{|x|\sqrt{b^2-4ac}}, & \text{if } c<0, \end{cases}$$

71. $\displaystyle\int x^3\sqrt{x^2+a^2}\,dx = (\tfrac{1}{3}x^2-\tfrac{2}{15}a^2)(x^2+a^2)^{3/2},$

72. $\displaystyle\int x^n\sin(ax)\,dx = -\tfrac{1}{a}x^n\cos(ax) + \tfrac{n}{a}\int x^{n-1}\cos(ax)\,dx,$

73. $\displaystyle\int x^n\cos(ax)\,dx = \tfrac{1}{a}x^n\sin(ax) - \tfrac{n}{a}\int x^{n-1}\sin(ax)\,dx,$

74. $\displaystyle\int x^n e^{ax}\,dx = \frac{x^n e^{ax}}{a} - \tfrac{n}{a}\int x^{n-1}e^{ax}\,dx,$

75. $\displaystyle\int x^n\ln(ax)\,dx = x^{n+1}\left(\frac{\ln(ax)}{n+1} - \frac{1}{(n+1)^2}\right),$

76. $\displaystyle\int x^n(\ln ax)^m\,dx = \frac{x^{n+1}}{n+1}(\ln ax)^m - \frac{m}{n+1}\int x^n(\ln ax)^{m-1}\,dx.$

$$
\begin{aligned}
x^1 &= x^{\underline{1}} &&= x^{\overline{1}} \\
x^2 &= x^{\underline{2}} + x^{\underline{1}} &&= x^{\overline{2}} - x^{\overline{1}} \\
x^3 &= x^{\underline{3}} + 3x^{\underline{2}} + x^{\underline{1}} &&= x^{\overline{3}} - 3x^{\overline{2}} + x^{\overline{1}} \\
x^4 &= x^{\underline{4}} + 6x^{\underline{3}} + 7x^{\underline{2}} + x^{\underline{1}} &&= x^{\overline{4}} - 6x^{\overline{3}} + 7x^{\overline{2}} - x^{\overline{1}} \\
x^5 &= x^{\underline{5}} + 15x^{\underline{4}} + 25x^{\underline{3}} + 10x^{\underline{2}} + x^{\underline{1}} &&= x^{\underline{5}} - 15x^{\overline{4}} + 25x^{\overline{3}} - 10x^{\overline{2}} + x^{\overline{1}}
\end{aligned}
$$

$$
\begin{aligned}
x^{\overline{1}} &= x^1 & x^{\underline{1}} &= x^1 \\
x^{\overline{2}} &= x^2 + x^1 & x^{\underline{2}} &= x^2 - x^1 \\
x^{\overline{3}} &= x^3 + 3x^2 + 2x^1 & x^{\underline{3}} &= x^3 - 3x^2 + 2x^1 \\
x^{\overline{4}} &= x^4 + 6x^3 + 11x^2 + 6x^1 & x^{\underline{4}} &= x^4 - 6x^3 + 11x^2 - 6x^1 \\
x^{\overline{5}} &= x^5 + 10x^4 + 35x^3 + 50x^2 + 24x^1 & x^{\underline{5}} &= x^5 - 10x^4 + 35x^3 - 50x^2 + 24x^1
\end{aligned}
$$

Finite Calculus

Difference, shift operators:

$$\Delta f(x) = f(x+1) - f(x),$$
$$\mathrm{E}\, f(x) = f(x+1).$$

Fundamental Theorem:

$$f(x) = \Delta F(x) \Leftrightarrow \sum f(x)\delta x = F(x) + C.$$
$$\sum_a^b f(x)\delta x = \sum_{i=a}^{b-1} f(i).$$

Differences:

$\Delta(cu) = c\Delta u, \qquad \Delta(u+v) = \Delta u + \Delta v,$

$\Delta(uv) = u\Delta v + \mathrm{E}\, v\Delta u,$

$\Delta(x^{\underline{n}}) = nx^{\underline{n-1}},$

$\Delta(H_x) = x^{\underline{-1}}, \qquad \Delta(2^x) = 2^x,$

$\Delta(c^x) = (c-1)c^x, \qquad \Delta\binom{x}{m} = \binom{x}{m-1}.$

Sums:

$\sum cu\,\delta x = c\sum u\,\delta x,$

$\sum (u+v)\,\delta x = \sum u\,\delta x + \sum v\,\delta x,$

$\sum u\Delta v\,\delta x = uv - \sum \mathrm{E}\, v\Delta u\,\delta x,$

$\sum x^{\underline{n}}\delta x = \frac{x^{\underline{n+1}}}{m+1}, \qquad \sum x^{\underline{-1}}\delta x = H_x,$

$\sum c^x\,\delta x = \frac{c^x}{c-1}, \qquad \sum \binom{x}{m}\delta x = \binom{x}{m+1}.$

Falling Factorial Powers:

$$x^{\underline{n}} = x(x-1)\cdots(x-n+1), \quad n>0,$$
$$x^{\underline{0}} = 1,$$
$$x^{\underline{n}} = \frac{1}{(x+1)\cdots(x+|n|)}, \quad n<0,$$
$$x^{\underline{n+m}} = x^{\underline{m}}(x-m)^{\underline{n}}.$$

Rising Factorial Powers:

$$x^{\overline{n}} = x(x+1)\cdots(x+n-1), \quad n>0,$$
$$x^{\overline{0}} = 1,$$
$$x^{\overline{n}} = \frac{1}{(x-1)\cdots(x-|n|)}, \quad n<0,$$
$$x^{\overline{n+m}} = x^{\overline{m}}(x+m)^{\overline{n}}.$$

Conversion:

$$x^{\underline{n}} = (-1)^n(-x)^{\overline{n}} = (x-n+1)^{\overline{n}} = 1/(x+1)^{\overline{-n}},$$
$$x^{\overline{n}} = (-1)^n(-x)^{\underline{n}} = (x+n-1)^{\underline{n}} = 1/(x-1)^{\underline{-n}},$$
$$x^n = \sum_{k=1}^n \left\{ n \atop k \right\} x^{\underline{k}} = \sum_{k=1}^n \left\{ n \atop k \right\} (-1)^{n-k} x^{\overline{k}},$$
$$x^{\underline{n}} = \sum_{k=1}^n \left[n \atop k \right] (-1)^{n-k} x^k,$$
$$x^{\overline{n}} = \sum_{k=1}^n \left[n \atop k \right] x^k.$$

Theoretical Computer Science Cheat Sheet

Series

Taylor's series:

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2}f''(a) + \cdots = \sum_{i=0}^{\infty} \frac{(x-a)^i}{i!} f^{(i)}(a).$$

Expansions:

$$\begin{aligned}
\frac{1}{1-x} &= 1 + x + x^2 + x^3 + x^4 + \cdots &&= \sum_{i=0}^{\infty} x^i, \\
\frac{1}{1-cx} &= 1 + cx + c^2x^2 + c^3x^3 + \cdots &&= \sum_{i=0}^{\infty} c^i x^i, \\
\frac{1}{1-x^n} &= 1 + x^n + x^{2n} + x^{3n} + \cdots &&= \sum_{i=0}^{\infty} x^{ni}, \\
\frac{x}{(1-x)^2} &= x + 2x^2 + 3x^3 + 4x^4 + \cdots &&= \sum_{i=0}^{\infty} ix^i, \\
x^k \frac{d^n}{dx^n}\left(\frac{1}{1-x}\right) &= x + 2^n x^2 + 3^n x^3 + 4^n x^4 + \cdots &&= \sum_{i=0}^{\infty} i^n x^i, \\
e^x &= 1 + x + \tfrac{1}{2}x^2 + \tfrac{1}{6}x^3 + \cdots &&= \sum_{i=0}^{\infty} \frac{x^i}{i!}, \\
\ln(1+x) &= x - \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 - \tfrac{1}{4}x^4 - \cdots &&= \sum_{i=1}^{\infty} (-1)^{i+1}\frac{x^i}{i}, \\
\ln\frac{1}{1-x} &= x + \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 + \tfrac{1}{4}x^4 + \cdots &&= \sum_{i=1}^{\infty} \frac{x^i}{i}, \\
\sin x &= x - \tfrac{1}{3!}x^3 + \tfrac{1}{5!}x^5 - \tfrac{1}{7!}x^7 + \cdots &&= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!}, \\
\cos x &= 1 - \tfrac{1}{2!}x^2 + \tfrac{1}{4!}x^4 - \tfrac{1}{6!}x^6 + \cdots &&= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!}, \\
\tan^{-1} x &= x - \tfrac{1}{3}x^3 + \tfrac{1}{5}x^5 - \tfrac{1}{7}x^7 + \cdots &&= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)}, \\
(1+x)^n &= 1 + nx + \tfrac{n(n-1)}{2}x^2 + \cdots &&= \sum_{i=0}^{\infty} \binom{n}{i} x^i, \\
\frac{1}{(1-x)^{n+1}} &= 1 + (n+1)x + \tbinom{n+2}{2}x^2 + \cdots &&= \sum_{i=0}^{\infty} \binom{i+n}{i} x^i, \\
\frac{x}{e^x - 1} &= 1 - \tfrac{1}{2}x + \tfrac{1}{12}x^2 - \tfrac{1}{720}x^4 + \cdots &&= \sum_{i=0}^{\infty} \frac{B_i x^i}{i!}, \\
\frac{1}{2x}(1 - \sqrt{1-4x}) &= 1 + x + 2x^2 + 5x^3 + \cdots &&= \sum_{i=0}^{\infty} \frac{1}{i+1}\binom{2i}{i} x^i, \\
\frac{1}{\sqrt{1-4x}} &= 1 + x + 2x^2 + 6x^3 + \cdots &&= \sum_{i=0}^{\infty} \binom{2i}{i} x^i, \\
\frac{1}{\sqrt{1-4x}}\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n &= 1 + (2+n)x + \tbinom{4+n}{2}x^2 + \cdots &&= \sum_{i=0}^{\infty} \binom{2i+n}{i} x^i, \\
\frac{1}{1-x}\ln\frac{1}{1-x} &= x + \tfrac{3}{2}x^2 + \tfrac{11}{6}x^3 + \tfrac{25}{12}x^4 + \cdots &&= \sum_{i=1}^{\infty} H_i x^i, \\
\frac{1}{2}\left(\ln\frac{1}{1-x}\right)^2 &= \tfrac{1}{2}x^2 + \tfrac{3}{4}x^3 + \tfrac{11}{24}x^4 + \cdots &&= \sum_{i=2}^{\infty} \frac{H_{i-1}x^i}{i}, \\
\frac{x}{1-x-x^2} &= x + x^2 + 2x^3 + 3x^4 + \cdots &&= \sum_{i=0}^{\infty} F_i x^i, \\
\frac{F_n x}{1-(F_{n-1}+F_{n+1})x-(-1)^n x^2} &= F_n x + F_{2n}x^2 + F_{3n}x^3 + \cdots &&= \sum_{i=0}^{\infty} F_{ni} x^i.
\end{aligned}$$

Ordinary power series:

$$A(x) = \sum_{i=0}^{\infty} a_i x^i.$$

Exponential power series:

$$A(x) = \sum_{i=0}^{\infty} a_i \frac{x^i}{i!}.$$

Dirichlet power series:

$$A(x) = \sum_{i=1}^{\infty} \frac{a_i}{i^x}.$$

Binomial theorem:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k.$$

Difference of like powers:

$$x^n - y^n = (x-y)\sum_{k=0}^{n-1} x^{n-1-k} y^k.$$

For ordinary power series:

$$\begin{aligned}
\alpha A(x) + \beta B(x) &= \sum_{i=0}^{\infty} (\alpha a_i + \beta b_i) x^i, \\
x^k A(x) &= \sum_{i=k}^{\infty} a_{i-k} x^i, \\
\frac{A(x) - \sum_{i=0}^{k-1} a_i x^i}{x^k} &= \sum_{i=0}^{\infty} a_{i+k} x^i, \\
A(cx) &= \sum_{i=0}^{\infty} c^i a_i x^i, \\
A'(x) &= \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i, \\
xA'(x) &= \sum_{i=1}^{\infty} i a_i x^i, \\
\int A(x)\,dx &= \sum_{i=1}^{\infty} \frac{a_{i-1}}{i} x^i, \\
\frac{A(x) + A(-x)}{2} &= \sum_{i=0}^{\infty} a_{2i} x^{2i}, \\
\frac{A(x) - A(-x)}{2} &= \sum_{i=0}^{\infty} a_{2i+1} x^{2i+1}.
\end{aligned}$$

Summation: If $b_i = \sum_{j=0}^{i} a_i$ then

$$B(x) = \frac{1}{1-x}A(x).$$

Convolution:

$$A(x)B(x) = \sum_{i=0}^{\infty} \left(\sum_{j=0}^{i} a_j b_{i-j}\right) x^i.$$

God made the natural numbers;
all the rest is the work of man.
– Leopold Kronecker

Theoretical Computer Science Cheat Sheet

Series

Expansions:

$$\frac{1}{(1-x)^{n+1}}\ln\frac{1}{1-x} = \sum_{i=0}^{\infty}(H_{n+i}-H_n)\binom{n+i}{i}x^i, \qquad \left(\frac{1}{x}\right)^{\overline{-n}} = \sum_{i=0}^{\infty}\left\{ {i \atop n} \right\}x^i,$$

$$x^{\overline{n}} = \sum_{i=0}^{\infty}\left[{n \atop i} \right]x^i, \qquad (e^x-1)^n = \sum_{i=0}^{\infty}\left\{ {i \atop n} \right\}\frac{n!x^i}{i!},$$

$$\left(\ln\frac{1}{1-x}\right)^n = \sum_{i=0}^{\infty}\left[{i \atop n} \right]\frac{n!x^i}{i!}, \qquad x\cot x = \sum_{i=0}^{\infty}\frac{(-4)^i B_{2i}x^{2i}}{(2i)!},$$

$$\tan x = \sum_{i=1}^{\infty}(-1)^{i-1}\frac{2^{2i}(2^{2i}-1)B_{2i}x^{2i-1}}{(2i)!}, \qquad \zeta(x) = \sum_{i=1}^{\infty}\frac{1}{i^x},$$

$$\frac{1}{\zeta(x)} = \sum_{i=1}^{\infty}\frac{\mu(i)}{i^x}, \qquad \frac{\zeta(x-1)}{\zeta(x)} = \sum_{i=1}^{\infty}\frac{\phi(i)}{i^x},$$

$$\zeta(x) = \prod_p \frac{1}{1-p^{-x}},$$

$$\zeta^2(x) = \sum_{i=1}^{\infty}\frac{d(i)}{x^i} \quad \text{where } d(n)=\textstyle\sum_{d|n}1,$$

$$\zeta(x)\zeta(x-1) = \sum_{i=1}^{\infty}\frac{S(i)}{x^i} \quad \text{where } S(n)=\textstyle\sum_{d|n}d,$$

$$\zeta(2n) = \frac{2^{2n-1}|B_{2n}|}{(2n)!}\pi^{2n}, \quad n\in\mathbb{N},$$

$$\frac{x}{\sin x} = \sum_{i=0}^{\infty}(-1)^{i-1}\frac{(4^i-2)B_{2i}x^{2i}}{(2i)!},$$

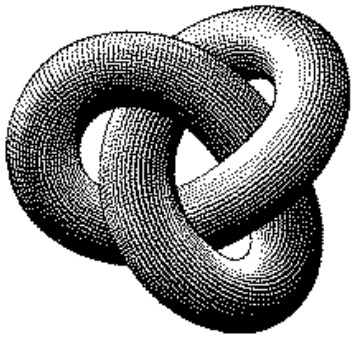
$$\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = \sum_{i=0}^{\infty}\frac{n(2i+n-1)!}{i!(n+i)!}x^i,$$

$$e^x\sin x = \sum_{i=1}^{\infty}\frac{2^{i/2}\sin\frac{i\pi}{4}}{i!}x^i,$$

$$\sqrt{\frac{1-\sqrt{1-x}}{x}} = \sum_{i=0}^{\infty}\frac{(4i)!}{16^i\sqrt{2}(2i)!(2i+1)!}x^i,$$

$$\left(\frac{\arcsin x}{x}\right)^2 = \sum_{i=0}^{\infty}\frac{4^i i!^2}{(i+1)(2i+1)!}x^{2i}.$$

Escher's Knot



Stieltjes Integration

If G is continuous in the interval $[a,b]$ and F is nondecreasing then

$$\int_a^b G(x)\,dF(x)$$

exists. If $a\le b\le c$ then

$$\int_a^c G(x)\,dF(x) = \int_a^b G(x)\,dF(x) + \int_b^c G(x)\,dF(x).$$

If the integrals involved exist

$$\int_a^b \big(G(x)+H(x)\big)\,dF(x) = \int_a^b G(x)\,dF(x) + \int_a^b H(x)\,dF(x),$$

$$\int_a^b G(x)\,d\big(F(x)+H(x)\big) = \int_a^b G(x)\,dF(x) + \int_a^b G(x)\,dH(x),$$

$$\int_a^b c\cdot G(x)\,dF(x) = \int_a^b G(x)\,d\big(c\cdot F(x)\big) = c\int_a^b G(x)\,dF(x),$$

$$\int_a^b G(x)\,dF(x) = G(b)F(b) - G(a)F(a) - \int_a^b F(x)\,dG(x).$$

If the integrals involved exist, and F possesses a derivative F' at every point in $[a,b]$ then

$$\int_a^b G(x)\,dF(x) = \int_a^b G(x)F'(x)\,dx.$$

Cramer's Rule

If we have equations:

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2 \\ \vdots \qquad\quad & \quad \vdots \qquad\quad \vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n &= b_n \end{aligned}$$

Let $A=(a_{i,j})$ and B be the column matrix (b_i). Then there is a unique solution iff $\det A\neq 0$. Let A_i be A with column i replaced by B. Then

$$x_i = \frac{\det A_i}{\det A}.$$

Improvement makes strait roads, but the crooked roads without Improvement, are roads of Genius.
– William Blake (The Marriage of Heaven and Hell)

00	47	18	76	29	93	85	34	61	52
86	11	57	28	70	39	94	45	02	63
95	80	22	67	38	71	49	56	13	04
59	96	81	33	07	48	72	60	24	15
73	69	90	82	44	17	58	01	35	26
68	74	09	91	83	55	27	12	46	30
37	08	75	19	92	84	66	23	50	41
14	25	36	40	51	62	03	77	88	99
21	32	43	54	65	06	10	89	97	78
42	53	64	05	16	20	31	98	79	87

The Fibonacci number system: Every integer n has a unique representation

$$n = F_{k_1} + F_{k_2} + \cdots + F_{k_m},$$

where $k_i \ge k_{i+1}+2$ for all i, $1\le i<m$ and $k_m\ge 2$.

Fibonacci Numbers

$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$

Definitions:

$$F_i = F_{i-1}+F_{i-2}, \quad F_0 = F_1 = 1,$$

$$F_{-i} = (-1)^{i-1}F_i,$$

$$F_i = \tfrac{1}{\sqrt{5}}\left(\phi^i - \hat{\phi}^i\right),$$

Cassini's identity: for $i>0$:

$$F_{i+1}F_{i-1} - F_i^2 = (-1)^i.$$

Additive rule:

$$F_{n+k} = F_kF_{n+1} + F_{k-1}F_n,$$

$$F_{2n} = F_nF_{n+1} + F_{n-1}F_n.$$

Calculation by matrices:

$$\begin{pmatrix} F_{n-2} & F_{n-1} \\ F_{n-1} & F_n \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n.$$