第五-七章 目录

第五章 博弈论

1、Wythoff(黄金分割，两堆石头，可以一起拿一样的)

```
/*
有 2 堆石子。A B 两个人轮流拿，A 先拿。每次可以从一堆中取任意个或从
2 堆中取相同数量的石子，但不可不取。拿到最后 1 颗石子的人获胜。假设
A B 都非常聪明，拿石子的过程中不会出现失误。给出 2 堆石子的数量，问
最后谁能赢得比赛。
*/
int main()
{
    int t, a, b, m, k;
    scanf("%d", &t);
    while (t--)
    {
        scanf("%d%d", &a, &b);
        if (a > b)
        {
            a ^= b;
            b ^= a;
            a ^= b;
        }
        m = b - a;
        k = (int)(m * (1 + sqrt(5)) / 2.0);
        //m = ? * a
        //k = m / ?
        //?:黄金分割数
        //如果 a == k，则为后手赢，否则先手赢（奇异局）
        printf("%s\n", a == k ? "B" : "A");
    }
    return 0;
}
```

2、Bash(石头谁先拿到最后一个)

```
/*
有一堆石子共有 N 个。A B 两个人轮流拿，A 先拿。每次最少拿 1 颗，
最多拿 K 颗，拿到最后 1 颗石子的人获胜。假设 A B 都非常聪明，
拿石子的过程中不会出现失误。给出 N 和 K，问最后谁能赢得比赛。
*/
/*    bashgame        */
int bash(int N, int K)
{
```

```
    if (N % (K + 1) == 0)      return 2;
    return 1;
}
```

3、nim 游戏，N 堆石子，每次可全拿

/*有 N 堆石子。A B 两个人轮流拿，A 先拿。每次只能从一堆中取若干个，可将一堆全取走，但不可不取，拿到最后 1 颗石子的人获胜。假设 A B 都非常聪明，拿石子的过程中不会出现失误。给出 N 及每堆石子的数量，问最后谁能赢得比赛。*/

```
/*Nim  游戏*/
int main(int argc, const char * argv[])
{
    int N, stone, tag = 0;
    scanf("%d", &N);
    while (N--){
        scanf("%d", &stone);
        tag ^= stone;
    }
    //tag 为 0 则为后手赢，否则为先手赢
    printf("%c\n", tag == 0 ? 'B' : 'A');
    return 0;
}
```

4、sg 函数

```
const int MAX_DIG = 64;
//f[]：可以取走的石子个数
//sg[]:0~n 的 SG 函数值
//hash[]:mex{}
int f[MAX_DIG]={1,3},sg[MAX_DIG],hash[MAX_DIG];
int k;//k 是 f[]的有效长度
void getSG(int n)
{
        memset(sg,0,sizeof(sg));
        for(int i=1; i<=n; i++) {
                memset(hash,0,sizeof(hash));
                for(int j=0; f[j]<=i && j < k; j++) //k 是 f[]的有效长度
                        hash[sg[i-f[j]]]=1;
                for(int j=0; ; j++) {     //求 mes{}中未出现的最小的非负整数
                    if(hash[j]==0) {
                        sg[i]=j;
                        break;}
                 }
```

```
            }
}
int main(){
        int n;
        read(n);
        k=2;
        getSG(n);
        for(int i=0;i<=n;i++){
        cout<<sg[i]<<" ";
        }
        return 0;
}
```

第六章 计算几何

1、liuctic 计算几何库

```
/*   Liuctic 的计算几何库
 *   p-Lpoint ln, l - Lline ls - Llineseglr - Lrad
 *   平面上两点之间的距离                    p2pdis
 *   (P1-P0)*(P2-P0)的叉积                xmulti
 *   确定两条线段是否相交                    lsinterls
 *   判断点 p 是否在线段 l 上                  ponls
 *   判断两个点是否相等                      Euqal_Point
 *   线段非端点相交                        lsinterls_A
 *   判断点 q 是否在多边形 Polygon 内          pinplg
 *   多边形的面积                         area_of_polygon
 *   解二次方程 Ax^2+Bx+C=0               equa
 *   计算直线的一般式 Ax+By+C=0            format
 *   点到直线距离                         p2lndis
 *   直线与圆的交点,已知直线与圆相交          lncrossc
 *   点是否在射线的正向                     samedir
 *   射线与圆的第一个交点                    lrcrossc
 *   求点 p1 关于直线 ln 的对称点 p2           mirror
 *   两直线夹角(弧度)                      angle_LL
 *   求两圆相交的面积                      Area_of_overlap
 *   求两矩形相交的面积                     Area_of_overlap_rec
 */
#define infinity 1e20
#define EP 1e-10
const int MAXV = 300;
const double PI = 2.0 * asin(1.0);   //   高精度 PI
struct Lpoint{   double x, y;}; //   点
struct Llineseg{     Lpoint a, b;}; //   线段
```

```
struct Ldir{    double dx, dy;};    //    方向向量
struct Lline{    Lpoint p;      Ldir dir;};    //    直线
struct Lrad{    Lpoint Sp;    Ldir dir;};    //    射线
struct Lround{      Lpoint co; double r;}; //    圆
double p2pdis(Lpoint p1, Lpoint p2){
      return (sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y)));
}
/*    若结果为正,则<P0,P1>在<P0,P2>的顺时针方向;
 *    若为 0 则<P0,P1><P0,P2>共线;
 *    若为负则<P0,P1>在<P0,P2>的在逆时针方向;
 *    可以根据这个函数确定两条线段在交点处的转向,比如确定 p0p1 和 p1p2 在 p1 处是左转还是右转,只要求(p2-p0)*(p1-p0),
 *    若<0 则左转,>0 则右转,=0 则共线*/
double xmulti(Lpoint p1, Lpoint p2, Lpoint p0){
      return ((p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y));
}
//确定两条线段是否相交
double mx(double t1, double t2){
      if (t1 > t2)       return t1;
      return t2;
}
double mn(double t1, double t2){
      if (t1 < t2)       return t1;
      return t2;
}
int lsinterls(Llineseg u, Llineseg v){
      return ((mx(u.a.x, u.b.x) >= mn(v.a.x, v.b.x)) && (mx(v.a.x, v.b.x) >= mn(u.a.x, u.b.x)) && (mx(u.a.y, u.b.y) >= mn(v.a.y, v.b.y)) && (mx(v.a.y, v.b.y) >= mn(u.a.y, u.b.y)) && (xmulti(v.a, u.b, u.a) * xmulti(u.b, v.b, u.a) >= 0) && (xmulti(u.a, v.b, v.a) * xmulti(v.b, u.b, v.a) >= 0));
}
//判断两个点是否相等
int Euqal_Point(Lpoint p1, Lpoint p2){
      return ((fabs(p1.x - p2.x) < EP) && (fabs(p1.y - p2.y) < EP));
}
//判断点 p 是否在线段 l 上
int ponls(Llineseg l, Lpoint p){
      return ((xmulti(l.b, p, l.a) == 0) && (((p.x - l.a.x) * (p.x - l.b.x) < 0) || ((p.y - l.a.y) * (p.y - l.b.y) < 0)));
}
//线段相交判断函数
/*当且仅当 u,v 相交并且交点不是 u,v 的端点时函数为 true; */
int lsinterls_A(Llineseg u, Llineseg v){
      return ((lsinterls(u, v)) && (!Euqal_Point(u.a, v.a)) && (!Euqal_Point(u.a, v.b)) && (!Euqal_Point(u.b, v.a)) && (!Euqal_Point(u.b, v.b)));
}
```

```
//判断点 q 是否在多边形内
/*   其中多边形是任意的凸或凹多边形,
 *   Polygon 中存放多边形的逆时针顶点序列 */
int pinplg(int vcount, Lpoint Polygon[], Lpoint q){
    int c = 0, i, n;
    Llineseg l1, l2;
    l1.a = q;
    l1.b = q;
    l1.b.x = infinity;
    n = vcount;
    for (i = 0; i < vcount; i++){
        l2.a = Polygon[i];
        l2.b = Polygon[(i + 1) % n];
        if ((lsinterls_A(l1, l2)) || ((ponls(l1, Polygon[(i + 1) % n])) && (((!ponls(l1, Polygon[(i + 2) % n]))
&& (xmulti(Polygon[i], Polygon[(i + 1) % n], l1.a) * xmulti(Polygon[(i + 1) % n], Polygon[(i + 2) % n], l1.a) >
0)) || ((ponls(l1, Polygon[(i + 2) % n])) && (xmulti(Polygon[i], Polygon[(i + 2) % n], l1.a) *
xmulti(Polygon[(i + 2) % n], Polygon[(i + 3) % n], l1.a) > 0)))))
            c++;
    }
    return (c % 2 != 0);
}
//多边形的面积
/* 要求按照逆时针方向输入多边形顶点
 *   可以是凸多边形或凹多边形 */
double area_of_polygon(int vcount, Lpoint plg[]){
    int i;
    double s;
    if (vcount < 3)       return 0;
    s = plg[0].y * (plg[vcount - 1].x - plg[1].x);
    for (i = 1; i < vcount; i++){
        s += plg[i].y * (plg[(i - 1)].x - plg[(i + 1) % vcount].x);
    }
    return s / 2;
}
//解二次方程  Ax^2+Bx+C=0
/*返回-1 表示无解  返回 1  表示有解*/
int equa(double A, double B, double C, double &x1, double &x2){
    double f = B * B - 4 * A * C;
    if (f < 0)       return -1;
    x1 = (-B + sqrt(f)) / (2 * A);
    x2 = (-B - sqrt(f)) / (2 * A);
    return 1;
}
//计算直线的一般式  Ax+By+C=0
```

```
void format(Lline ln, double &A, double &B, double &C){
	A = ln.dir.dy;
	B = -ln.dir.dx;
	C = ln.p.y * ln.dir.dx - ln.p.x * ln.dir.dy;
	return ;
}
//点到直线距离
double p2lndis(Lpoint a, Lline ln){
	double A, B, C;
	format(ln, A, B, C);
	return (fabs(A * a.x + B * a.y + C) / sqrt(A * A + B * B));
}
//直线与圆的交点,已知直线与圆相交
int lncrossc(Lline ln, Lround Y, Lpoint &p1, Lpoint &p2){
	double A, B, C, t1, t2;
	int zz = -1;
	format(ln, A, B, C);
	if (fabs(B) < 1e-8){
		p1.x = p2.x = -1.0 * C / A;
		zz = equa(1.0, -2.0 * Y.co.y, Y.co.y * Y.co.y + (p1.x - Y.co.x) * (p1.x - Y.co.x) - Y.r * Y.r, t1, t2);
		p1.y = t1;
		p2.y = t2;
	}
	else if (fabs(A) < 1e-8){
		p1.y = p2.y = -1.0 * C / B;
		zz = equa(1.0, -2.0 * Y.co.x, Y.co.x * Y.co.x + (p1.y - Y.co.y) * (p1.y - Y.co.y) - Y.r * Y.r, t1, t2);
		p1.x = t1;
		p2.x = t2;
	}
	else{
		zz = equa(A * A + B * B, 2.0 * A * C + 2.0 * A * B * Y.co.y - 2.0 * B * B * Y.co.x, B * B * Y.co.x *
Y.co.x + C * C + 2* B * C * Y.co.y + B * B * Y.co.y * Y.co.y - B * B * Y.r * Y.r, t1, t2);
		p1.x = t1, p1.y = -1 * (A / B * t1 + C / B);
		p2.x = t2, p2.y = -1 * (A / B * t2 + C / B);
	}
	return 0;
}
//点是否在射线的正向
bool samedir(Lrad ln, Lpoint P){
	double ddx, ddy;
	ddx = P.x - ln.Sp.x;
	ddy = P.y - ln.Sp.y;
	if ((ddx * ln.dir.dx > 0 || fabs(ddx * ln.dir.dx) < 1e-7) && (ddy * ln.dir.dy > 0 || (fabs(ddy * ln.dir.dy)
< 1e-7)))    return true;
```

```
    else      return false;
}
//射线与圆的第一个交点
/*  已经确定射线所在直线与圆相交返回-1 表示不存正向交点,否则返回 1 */
int lrcrossc(Lrad ln, Lround Y, Lpoint &P){
    Lline ln2;
    Lpoint p1, p2;
    int res = -1;
    double dis = 1e20;
    ln2.p = ln.Sp, ln2.dir = ln.dir;
    lncrossc(ln2, Y, p1, p2);
    if (samedir(ln, p1)){
        res = 1;
        if (p2pdis(p1, ln.Sp) < dis)        dis = p2pdis(p1, ln.Sp);
        P = p1;
    }
    if (samedir(ln, p2)){
        res = 1;
        if (p2pdis(p2, ln.Sp) < dis){
            dis = p2pdis(p2, ln.Sp);
            P = p2;
        }
    }
    return res;
}
//求点 p1 关于直线 ln 的对称点 p2
Lpoint mirror(Lpoint P, Lline ln){
    Lpoint Q;
    double A, B, C;
    format(ln, A, B, C);
    Q.x = ((B * B - A * A) * P.x - 2 * A * B * P.y - 2 * A * C) / (A * A + B * B);
    Q.y = ((A * A - B * B) * P.y - 2 * A * B * P.x - 2 * B * C) / (A * A + B * B);
    return Q;
}
//两直线夹角(弧度)
double angle_LL(Lline line1, Lline line2){
    double A1, B1, C1;
    format(line1, A1, B1, C1);
    double A2, B2, C2;
    format(line2, A2, B2, C2);
    if (A1 * A2 + B1 * B2 == 0)        return PI / 2.0;        //    垂直
    else{
        double t = fabs((A1 * B2 - A2 * B1) / (A1 * A2 + B1 * B2));
        return atan(t);
```

```
    }
}
//求两圆相交的面积
double Area_of_overlap(Point c1, double r1, Point c2, double r2) {
    double d = dist(c1, c2);
    if (r1 + r2 < d + eps)      return 0;
    if (d < fabs(r1 - r2) + eps){
        double r = min(r1, r2);
        return PI * r * r;
    }
    double x = (d * d + r1 * r1 - r2 * r2) / (2 * d);
    double t1 = acos(x / r1);
    double t2 = acos((d - x) / r2);
    return r1 * r1 * t1 + r2 * r2 * t2 - d * r1 * sin(t1);
}
/* x[]、y[]存储矩阵对角线顶点（只需要任意一条） */
double Area_of_overlap_rec(double x[], double y[]){
    //  将两个矩形全部统一为主对角线
    sort(x, x + 2);
    sort(x + 2, x + 4);
    sort(y , y + 2);
    sort(y + 2, y + 4);
    if (x[1] <= x[2] || x[0] >= x[3] || y[0] >= y[3] || y[1] <= y[2])      return 0.0;    //  相离
    else{
        sort(x, x + 4);
        sort(y, y + 4);
        return (x[2] - x[1]) * (y[2] - y[1]);
    }
}
```

2、判断多边形重心

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

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

3、判断两线段相交

```
const double eps = 1e-10;
struct point{ double x, y;};
double min(double a, double b){ return a < b ? a : b;}
double max(double a, double b){    return a > b ? a : b;}
bool inter(point a, point b, point c, point d){
        if (min(a.x, b.x) > max(c.x, d.x) || min(a.y, b.y) > max(c.y, d.y) || min(c.x, d.x) > max(a.x, b.x)
|| min(c.y, d.y) > max(a.y, b.y)){
            return 0;
        }
        double h, i, j, k;
        h = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
        i = (b.x - a.x) * (d.y - a.y) - (b.y - a.y) * (d.x - a.x);
        j = (d.x - c.x) * (a.y - c.y) - (d.y - c.y) * (a.x - c.x);
        k = (d.x - c.x) * (b.y - c.y) - (d.y - c.y) * (b.x - c.x);
        return h * i <= eps && j * k <= eps;
}
```

4、判断四点共面

```
struct    point{
        double x, y, z;
        point    operator - (point &o){
            point    ans;
            ans.x = this->x - o.x;
            ans.y = this->y - o.y;
            ans.z = this->z - o.z;
            return ans;
        }
};
double    dot_product(const point &a, const point &b){
```

```
    return a.x * b.x + a.y * b.y + a.z * b.z;
}
point cross_product(const point &a, const point &b){
    point    ans;
    ans.x = a.y * b.z - a.z * b.y;
    ans.y = a.z * b.x - a.x * b.z;
    ans.z = a.x * b.y - a.y * b.x;
    return ans;
}
int main(){
    point p[4];
    int T;
    for (scanf("%d", &T); T--;){
        for (int i = 0; i < 4; ++i){
            scanf("%lf%lf%lf", &p[i].x, &p[i].y, &p[i].z);
        }
        puts(dot_product(p[3] - p[0], cross_product(p[2] - p[0], p[1] - p[0])) == 0.0 ? "Yes\n" :
"No\n");
    }
    return 0;
}
```

5、判断线段与圆是否相交

```
typedef struct { //    点结构
    ll x, y;
} Point;
Point A, B, C, O;     //    三角形三点与圆心
ll r;                          //    半径
//    判断线段是否和圆相交
int segOnCircle(Point *p_1, Point *p_2){
    ll a, b, c, dist_1, dist_2, angle_1, angle_2;     //    ax + by + c = 0;
    if (p_1->x == p_2->x){                                    //    当 x 相等
        a = 1, b = 0, c = -p_1->x;
    }
    else if (p_1->y == p_2->y){                              //    当 y 相等
        a = 0, b = 1, c = -p_1->y;
    }
    else{
        a = p_1->y - p_2->y;
        b = p_2->x - p_1->x;
        c = p_1->x * p_2->y - p_1->y * p_2->x;
    }
    dist_1 = a * O.x + b * O.y + c;
```

```
    dist_1 *= dist_1;
    dist_2 = (a * a + b * b) * r * r;
    if (dist_1 > dist_2)        return 0;
    angle_1 = (O.x - p_1->x) * (p_2->x - p_1->x) + (O.y - p_1->y) * (p_2->y - p_1->y);
    angle_2 = (O.x - p_2->x) * (p_1->x - p_2->x) + (O.y - p_2->y) * (p_1->y - p_2->y);
    if (angle_1 > 0 && angle_2 > 0)        return 1;
    return 0;
}
```

6、三角形

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

面积
三角形面积可以根据海伦公式求得：
s = sqrt(p * (p - a) * (p - b) * (p - c));
p = (a + b + c) / 2;

关键点与 A, B, C 三顶点距离之和
费马点
该点到三角形三个顶点的距离之和最小。
有个有趣的结论:
若三角形的三个内角均小于 120 度,那么该点连接三个顶点形成的三个角均为 120 度;若三角形存在一个内角大于 120 度,则该顶点就是费马点。
计算公式如下:
若有一个内角大于 120 度(这里假设为角 C),则距离为 a + b;若三个内角均小于 120 度,则距离为 sqrt((a * a + b * b + c * c + 4 * sqrt(3.0) * s) / 2)。

内心
角平分线的交点。
令 x = (a + b - c) / 2, y = (a - b + c) / 2, z = (-a + b + c) / 2, h = s / p.
计算公式为 sqrt(x * x + h * h) + sqrt(y * y + h * h) + sqrt(z * z + h * h)。

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

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

外心
三点求圆心坐标。

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

Pick 公式
顶点坐标均是整点的简单多边形:
面积 = 内部格点数目 + 边上格点数目 / 2 - 1

S = n + s / 2 - 1
(其中 n 表示多边形内部的点数,s 表示多边形边界上的点数,S 表示多边形的面积)

已知圆锥表面积 S 求最大体积 V
V = S * sqrt(S / (72 * Pi))\

第七章 二叉树及其他目录

1、中序遍历+后序遍历建二叉树

```
// UVa548 Tree
// Rujia Liu
// 题意：给一棵点带权（权各不相同，都是正整数）二叉树的中序和后序遍历，找一个叶子使得它到根的路径上的权和最小。如果有多解，该叶子本身的权应尽量小
// 算法：递归建树，然后 DFS。注意，直接递归求结果也可以，但是先建树的方法不仅直观，而且更好调试
// 因为各个结点的权值各不相同且都是正整数，直接用权值作为结点编号
const int maxv = 10000 + 10;
int in_order[maxv], post_order[maxv], lch[maxv], rch[maxv];
int n;
bool read_list(int* a) {
  string line;
  if(!getline(cin, line)) return false;
  stringstream ss(line);
  n = 0;
  int x;
  while(ss >> x) a[n++] = x;
  return n > 0;
```

```
}
// 把 in_order[L1..R1]和 post_order[L2..R2]建成一棵二叉树，返回树根
int build(int L1, int R1, int L2, int R2) {
   if(L1 > R1) return 0; // 空树
   int root = post_order[R2];
   int p = L1;
   while(in_order[p] != root) p++;
   int cnt = p-L1; // 左子树的结点个数
   lch[root] = build(L1, p-1, L2, L2+cnt-1);
   rch[root] = build(p+1, R1, L2+cnt, R2-1);
   return root;}
int best, best_sum; // 目前为止的最优解和对应的权和
void dfs(int u, int sum) {
   sum += u;
   if(!lch[u] && !rch[u]) // 叶子
      if(sum < best_sum || (sum == best_sum && u < best)) { best = u; best_sum =
sum; }
   if(lch[u]) dfs(lch[u], sum);
   if(rch[u]) dfs(rch[u], sum);
}
int main() {
   while(read_list(in_order)) {
      read_list(post_order);
      build(0, n-1, 0, n-1);
      best_sum = 1000000000;
      dfs(post_order[n-1], 0);
      cout << best << "\n";
   }
   return 0;
}
```

2、二叉树建树

```
// UVa122 Trees on the level
const int maxn = 256 + 10;
struct Node{
   bool have_value;
   int v;
   Node* left, *right;
   Node():have_value(false),left(NULL),right(NULL){}
};
Node* root;
Node* newnode() { return new Node(); }
bool failed;
```

```
void addnode(int v, char* s) {
    int n = strlen(s);
    Node* u = root;
    for(int i = 0; i < n; i++)
        if(s[i] == 'L') {
            if(u->left == NULL) u->left = newnode();
            u = u->left;
        } else if(s[i] == 'R') {
            if(u->right == NULL) u->right = newnode();
            u = u->right;
        }
    if(u->have_value) failed = true;
    u->v = v;
    u->have_value = true;
}
void remove_tree(Node* u) {
    if(u == NULL) return;
    remove_tree(u->left);
    remove_tree(u->right);
    delete u;      }
char s[maxn];
bool read_input() {
    failed = false;
    remove_tree(root);
    root = newnode();
    for(;;) {
        if(scanf("%s", s) != 1) return false;
        if(!strcmp(s, "()")) break;
        int v;
        sscanf(&s[1], "%d", &v);
        addnode(v, strchr(s, ',')+1);
    }
    return true; }
bool bfs(vector<int>& ans) {
    queue<Node*> q;
    ans.clear();
    q.push(root);
    while(!q.empty()) {
        Node* u = q.front(); q.pop();
        if(!u->have_value) return false;
        ans.push_back(u->v);
        if(u->left != NULL) q.push(u->left);
        if(u->right != NULL) q.push(u->right);
    }
```

```
    return true;    }
int main() {
    vector<int> ans;
    while(read_input()) {
        if(!bfs(ans)) failed = 1;
        if(failed) printf("not complete\n");
        else {
            for(int i = 0; i < ans.size(); i++) {
                if(i != 0) printf(" ");
                printf("%d", ans[i]);
            }
            printf("\n");
        }    }      return 0; }
```

3、C++大数

```
struct Bign
{
    int len,s[MAXN];
    Bign(){
        memset(s,0,sizeof(s));
        len=1;
    }
    Bign(int num){*this=num;}
    Bign(const char *num){*this=num;}
    void clean(){while(len>1&&!s[len-1])len--;}
    Bign operator = (const int num){
        char s[MAXN];
        sprintf(s,"%d",num);
        *this=s;
        return *this;
    }
    Bign operator = (const char *num) {
        len=strlen(num);
        for(int i=0;i<len;i++)s[i]=num[len-i-1]-'0';
        return *this;
    }
    Bign operator + (const Bign& b) {
        Bign c;
        c.len=0;
        for(int i=0,g=0;g||i<Max(len,b.len);i++){
            int x=g;
            if(i<b.len)x+=b.s[i];
```

```
            if(i<len)x+=s[i];
            c.s[c.len++]=x%10;
            g=x/10;
        }
        return c;
}
Bign operator - (const Bign& b){
        Bign c;
        c.len=0;
        for(int i=0,g=0;i<len;i++){
            int x=s[i]-g;
            if(i<b.len)x-=b.s[i];
            if(x>=0)g=0;
            else{g=1;x+=10;}
            c.s[c.len++]=x;
        }
        c.clean();
        return c;
}
Bign operator * (const Bign& b){
        Bign c;
        c.len=len+b.len;
        for(int i=0;i<len;i++){
            for(int j=0;j<b.len;j++)    c.s[i+j]+=s[i]*b.s[j];
        }
        for(int i=0;i<c.len;i++) {
            c.s[i+1]+=c.s[i]/10;
            c.s[i]%=10;
        }
        c.clean();
        return c;
}
Bign operator * (const int& b){
        Bign c;
        c.len=0;
        for(int i=0,g=0;g||i<len;i++){
            int x;
            if(i<len)x=s[i]*b+g;
            else x=g;
            c.s[c.len++]=x%10;
            g=x/10;
        }
        return c;
}
```

```
Bign operator / (const Bign& b){
	Bign c,f=0;
	for(int i=len-1;i>=0;i--){
		f=f*10;
		f.s[0]=s[i];
		while(f>=b){
			f=f-b;
			c.s[i]++;
		}
	}
	c.len=len;
	c.clean();
	return c;
}
Bign operator / (const int& b){
	Bign c,d=*this;
	c.len=len;
	for(int i=len-1,g=0;i>=0;i--){
		d.s[i]+=g*10;
		c.s[i]=d.s[i]/b;
		g=d.s[i]%b;
	}
	c.clean();
	return c;
}
Bign operator % (const Bign& b){
	Bign c=*this/b;
	c=*this-c*b;
	return c;
  }
Bign operator += (const Bign& b)
{*this=*this+b;return *this;}
Bign operator -= (const Bign& b)
{*this=*this-b;return *this;}
Bign operator *= (const Bign& b)
{*this=*this*b;return *this;}
Bign operator /= (const Bign& b)
{*this=*this/b;return *this;}
Bign operator *= (const int& b)
{*this=*this*b;return *this;}
Bign operator /= (const int& b)
{*this=*this/b;return *this;}
Bign operator %= (const Bign& b)
{*this=*this%b;return *this;}
```

```
    bool operator < (const Bign& b){
        if(b.len!=len)return len<b.len;
        for(int i=len-1;i>=0;i--)    if(s[i]!=b.s[i])return s[i]<b.s[i];
        return 0;
    }
    bool operator > (const Bign& b){
        if(b.len!=len)return len>b.len;
        for(int i=len-1;i>=0;i--)    if(s[i]!=b.s[i])return s[i]>b.s[i];
        return 0;
    }
    bool operator == (const Bign& b)
    {return !(*this>b)&&!(*this<b);}
    bool operator <= (const Bign& b)
    {return !(*this>b);}
    bool operator >= (const Bign& b)
    {return !(*this<b);}
    bool operator != (const Bign& b)
    {return !(*this==b);}
    string str() const {
        string res;
        for(int i=0;i<len;i++)
        res=char(s[i]+'0')+res;
        return res;
    }
};
//cin 读入
istream& operator >> (istream&in,Bign &x){
    string s;
    in>>s;
    x=s.c_str();
    return in;
 }
ostream& operator << (ostream&out,Bign x){
    out<<x.str();
    return out;
}
int main(){
    Bign a,b;
    cin>>a>>b;
    cout<<a%b<<endl;
    return 0;
}
```

4、基姆拉尔森公式,给年月日,计算星期几

```
W = (D + 2 * M + 3 * (M + 1) \ 5 + Y + Y \ 4 - Y \ 100 + Y \ 400) Mod 7;
```

5、java 大数

```
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.Scanner;
Scanner cin=new Scanner(System.in);
BigInteger num1=new BigInteger("12345");
BigInteger num2=cin.nextBigInteger();
BigDecimal num3=new BigDecimal("123.45");
BigDecimal num4=cin.nextBigDecimal();
BigInteger num1=new BigInteger("12345");
BigInteger num2=new BigInteger("45");
//加法
System.out.println(num1.add(num2));
//减法
System.out.println(num1.subtract(num2));
//乘法
System.out.println(num1.multiply(num2));
//除法(相除取整)
System.out.println(num1.divide(num2));
//取余
System.out.println(num1.mod(num2));
//最大公约数 GCD
System.out.println(num1.gcd(num2));
//取绝对值
System.out.println(num1.abs());
//取反
System.out.println(num1.negate());
//取最大值
System.out.println(num1.max(num2));
//取最小值
System.out.println(num1.min(num2));
//是否相等
System.out.println(num1.equals(num2));
BigDecimal num1=new BigDecimal("123.45");
BigDecimal num2=new BigDecimal("4.5");
//加法
System.out.println(num1.add(num2));
//减法
System.out.println(num1.subtract(num2));
```

```
//乘法
System.out.println(num1.multiply(num2));
//除法（在 divide 的时候就设置好要精确的小数位数和舍入模式）
System.out.println(num1.divide(num2,10,BigDecimal.ROUND_HALF_DOWN));
//取绝对值
System.out.println(num1.abs());
//取反
System.out.println(num1.negate());
//取最大值
System.out.println(num1.max(num2));
//取最小值
System.out.println(num1.min(num2));
//是否相等
System.out.println(num1.equals(num2));
//判断大小( >  返回 1, <  返回-1)
System.out.println(num2.compareTo(num1));
```

6、归并排序求逆序数

```
/*也可以用树状数组做
 *    a[0...n-1] cnt=0; call: MergeSort(0, n)*/
const int N = 1010;
int a[N],c[N],cnt=0;
void MergeSort(int l, int r){
    int mid, i, j, tmp;
    if (r > l + 1){
        mid = (l + r) / 2;
        MergeSort(l, mid);
        MergeSort(mid, r);
        tmp = l;
        for (i = l, j = mid; i < mid && j < r;){
            if (a[i] > a[j]){
                c[tmp++] = a[j++];
                cnt += mid - i;
            }
            else    c[tmp++] = a[i++];
        }
        if (j < r){
            for (; j < r; ++j)       c[tmp++] = a[j];
        }
        else{ for (; i < mid; ++i)       c[tmp++]=a[i];}
        for (i = l; i < r; ++i)      a[i] = c[i];
    }
    return ;}
```