Death_knight_DK 模板

常用定义

```
#define PI acos(-1.0)
#define e exp(1.0)
char *p = new char[10];    delete []p;

#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
```

判断 double 型 x 是否是整数的方法

```
zero(x-floor(x+eps))
```

不等式判断的话 x>=y 写成 x+eps>=y

优先队列：

```
struct thanks{
    friend bool operator< (thanks x, thanks y){
        return x.value > y.value;
    }
    int number;
    int value;
}f,c;
priority_queue <struct thanks>q;
Id=lower_bound(ls+1,ls+n+1,a[i])=ls;
```

图论

割点与割边

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<vector>
using namespace std;
struct bian{
    int end;
    int visit;
}w,z;
int vst[10005],level1[10005],level2[10005],l;
vector<struct bian> v[10005];
void dfs(int x,int y){
    vst[x]=1;
    level1[x]=++l;
    level2[x]=l;
    for(int i=0;i<v[x].size();i++){
        if(level2[v[x][i].end]==0){
            v[x][i].visit=1;
            dfs(v[x][i].end,x);
            if(level2[x]>level2[v[x][i].end]){

    level2[x]=level2[v[x][i].end];
            }
        }
        else{
            if(level1[v[x][i].end]<level2[x]
&& v[x][i].end!=y){

    level2[x]=level1[v[x][i].end];
            }
        }
    }
    return;
}
int main(){
    int i,j,k,m,n,a,b,h,max[10005],maxvalue;
    do{
        scanf("%d %d",&n,&m);
```

```
        if(n!=0){
            l=0;
            h=0;
            maxvalue=-1;
            memset(max,0,sizeof(max));
            memset(vst,0,sizeof(vst));

    memset(level1,0,sizeof(level1));

    memset(level2,0,sizeof(level2));
            for(i=0;i<=10000;i++){
                v[i].clear();
            }
            w.visit=0;
            z.visit=0;
            for(i=1;i<=m;i++){
                scanf("%d %d",&a,&b);
                w.end=b;
                z.end=a;
                v[a].push_back(w);
                v[b].push_back(z);
            }
            for(i=0;i<=n-1;i++){
                if(vst[i]==0){
                    h++;
                    max[i]--;
                    dfs(i,i);
                }
            }
            for(i=0;i<=n-1;i++){
                for(j=0;j<v[i].size();j++){

    if(level2[v[i][j].end]>=level1[i]          &&
v[i][j].visit==1){
                            max[i]++;
                    }
                }
                if(max[i]>maxvalue){
                    maxvalue=max[i];
                }
            }
            printf("%d\n",maxvalue+h);
        }
    }while(n!=0);
    return 0;
}
```

网络流 DINIC

DINIC 算法

```
const int maxnode = 1000 + 5;
const int maxedge = 1000 + 5;
const int oo = 1000000000;
int node, src, dest, nedge;
int head[maxnode], point[maxedge], next1[maxedge], flow[maxedge], capa[maxedge];//point[x]==y 表示第 x 条边连接 y，head，next 为邻接表，flow[x]表示 x 边的动态值，capa[x]表示 x 边的初始值
int dist[maxnode], Q[maxnode], work[maxnode];//dist[i]表示 i 点的等级
void init(int _node, int _src, int _dest){//初始化，node 表示点的个数，src 表示起点，dest 表示终点
    node = _node;
    src = _src;
    dest = _dest;
    for (int i = 0; i < node; i++) head[i] = -1;
    nedge = 0;
}
void addedge(int u, int v, int c1, int c2){//增加一条 u 到 v 流量为 c1，v 到 u 流量为 c2 的两条边
    point[nedge] = v, capa[nedge] = c1, flow[nedge] = 0, next1[nedge] = head[u], head[u] = (nedge++);
    point[nedge] = u, capa[nedge] = c2, flow[nedge] = 0, next1[nedge] = head[v], head[v] = (nedge++);
}
bool dinic_bfs(){
    memset(dist, 255, sizeof (dist));
    dist[src] = 0;
    int sizeQ = 0;
    Q[sizeQ++] = src;
    for (int cl = 0; cl < sizeQ; cl++)
        for (int k = Q[cl], i = head[k]; i >= 0; i = next1[i])
```

```
            if (flow[i] < capa[i] && dist[point[i]] < 0){
                dist[point[i]] = dist[k] + 1;
                Q[sizeQ++] = point[i];
            }
    return dist[dest] >= 0;
}
int dinic_dfs(int x, int exp){
    if (x == dest) return exp;
    for (int &i = work[x]; i >= 0; i = next1[i]){
        int v = point[i], tmp;
        if (flow[i] < capa[i] && dist[v] == dist[x] + 1 && (tmp = dinic_dfs(v, min(exp, capa[i] - flow[i]))) > 0){
            flow[i] += tmp;
            flow[i^1] -= tmp;
            return tmp;
        }
    }
    return 0;
}
int dinic_flow(){
    int result = 0;
    while (dinic_bfs()){
        for (int i = 0; i < node; i++) work[i] = head[i];
        while (1){
            int delta = dinic_dfs(src, oo);
            if (delta == 0) break;
            result += delta;
        }
    }
    return result;
}
//建图前,运行一遍 init()；
//加边时，运行 addedge(a,b,c,0),表示点 a 到 b 流量为 c 的边建成（注意点序号要从 0 开始）
//求解最大流运行 dinic_flow(),返回值即为答案
```

费用流

```
const int N = 1010;//点
const int M = 2 * 10010;//边
const int inf = 1000000000;
struct Node{//边，点 f 到点 t，流量为 c，费用为 w
    int f, t, c, w;
}e[M];
int next1[M], point[N], dis[N], q[N], pre[N], ne;//ne 为已添加的边数，next，point 为邻接表,dis 为花费，pre 为父亲节点
bool u[N];
void init(){
    memset(point, -1, sizeof(point));
    ne = 0;
}
void add_edge(int f, int t, int d1, int d2, int w){//f 到 t 的一条边，流量为 d1,反向流量 d2,花费 w,反向边花费-w（可以反悔）
    e[ne].f = f, e[ne].t = t, e[ne].c = d1, e[ne].w = w;
    next1[ne] = point[f], point[f] = ne++;
    e[ne].f = t, e[ne].t = f, e[ne].c = d2, e[ne].w = -w;
    next1[ne] = point[t], point[t] = ne++;
}
bool spfa(int s, int t, int n){
    int i, tmp, l, r;
    memset(pre, -1, sizeof(pre));
    for(i = 0; i < n; ++i)
        dis[i] = inf;
    dis[s] = 0;
    q[0] = s;
    l = 0, r = 1;
    u[s] = true;
    while(l != r) {
        tmp = q[l];
        l = (l + 1) % (n + 1);
        u[tmp] = false;
       for(i = point[tmp]; i != -1; i = next1[i])
{
```

```
                if(e[i].c && dis[e[i].t] > dis[tmp] + e[i].w) {
                    dis[e[i].t] = dis[tmp] + e[i].w;
                    pre[e[i].t] = i;
                    if(!u[e[i].t]) {
                        u[e[i].t] = true;
                        q[r] = e[i].t;
                        r = (r + 1) % (n + 1);
                    }
                }
            }
        }
        if(pre[t] == -1)
            return false;
        return true;
}
void MCMF(int s, int t, int n, int &flow, int &cost){//起点 s，终点 t，点数 n，最大流 flow，最小花费 cost
        int tmp, arg;
        flow = cost = 0;
        while(spfa(s, t, n)) {
            arg = inf, tmp = t;
            while(tmp != s) {
                arg = min(arg, e[pre[tmp]].c);
                tmp = e[pre[tmp]].f;
            }
            tmp = t;
            while(tmp != s) {
                e[pre[tmp]].c -= arg;
                e[pre[tmp] ^ 1].c += arg;
                tmp = e[pre[tmp]].f;
            }
            flow += arg;
            cost += arg * dis[t];
        }
}
//建图前运行 init()
//节点下标从 0 开始
//加边时运行 add_edge(a,b,c,0,d)表示加一条 a 到 b 的流量为 c 花费为 d 的边（注意花费为单位流量花费）
//特别注意双向边，如果要加的是双向边的话，运行 add_edge(a,b,c,0,d),add_edge(b,a,c,0,d) 较好，不要只运行一次 add_edge(a,b,c,c,d),费用会不对。
//求解时代入 MCMF(s,t,n,v1,v2)，表示起点为 s，终点为 t，点数为 n 的图中，最大流为 v1，最大花费为 v2
```

数据结构

并查集

```
int parent[];
int root(int p){
    if(parent[p]==-1) return p;
    else return parent[p]=root(parent[p]);
}
void merge(int a,int b){
    a=root(a);
    b=root(b);
    parent[a]=b;
}
```

哈希表

字符串处理哈希函数

```
unsigned int BKDRHash(char *str){
    unsigned int seed = 131; // 31 131 1313 13131 131313 etc..
    unsigned int hash = 0;
    while (*str){
        hash = hash * seed + (*str++);
    }
    return (hash & 0x7FFFFFFF);
}
```

树状数组

```
#include<cstdio>
#include<cstring>
int h[50005],n,o[50005],he2[50005];
```

```
int lowbit(int x){
	return x&(-x);
}
int getsum(int x){
	int sum=0;
	for(;x>0;x-=lowbit(x))
		sum+=h[x];
	return sum;
}
void update(int x,int v){
	for(;x<=n;x+=lowbit(x))
		h[x]+=v;
}
int main(){
	int i,j,m,a,b,k,c,l,T;
	char p[5];
	scanf("%d",&T);
	while(T--){
		scanf("%d%d",&n,&m);
		memset(h,0,sizeof(h));
		memset(he2,0,sizeof(he2));
		for(i=1;i<=n;i++){
			scanf("%d",&o[i]);
			he2[i]=he2[i-1]+o[i];

	h[i]=he2[i]-he2[i-lowbit(i)+1]+o[i-lowbit(i
)+1];
		}
			scanf("%d",&m);
		for(i=1;i<=m;i++){
			scanf("%s",p);
			if(p[0]=='a'){
				scanf("%d%d",&a,&b);
				update(a,b);
				o[a]+=b;
			}
			else{
				scanf("%d%d",&a,&b);
				l=o[a];
				l+=getsum(b)-getsum(a);
				printf("%d\n",l);
			}
		}
	}
	return 0;
}
```

求逆序数：

```
int main(){
	int
i,j,m,minzhi,maxzhi,xiao,da,dav,xiaoxuhao,dax
uhao,k,ans;
	scanf("%d",&n);
	for(i=1;i<=n;i++){
		scanf("%d",&a[i]);
		a[i]++;
	}
	minzhi=1+n;
	maxzhi=-1;
		memset(h,0,sizeof(h));
	for(i=1;i<=n;i++){
		update(a[i],1);
		nixu[i]=i-getsum(a[i]);
		}
}
```

线段树

```
#include<stdio.h>
#include<string.h>
struct Tree{
	int lson ,rson;
	int no;
}tree[5010*3];
int num[5010];
void build_tree(int id ,int left ,int right){
	tree[id].lson = left;
	tree[id].rson = right;
	tree[id].no = 0;
	if(left==right){
		return;
	}
	int mid = (left + right) / 2;
	build_tree(2*id,left,mid);
	build_tree(2*id+1,mid+1,right);
};
void update(int id ,int pos){
	if(tree[id].lson==tree[id].rson){
		tree[id].no++;
```

```
            return;
        }
        int mid = (tree[id].lson + tree[id].rson) /
2;
        if(pos <= mid){
            update(2*id,pos);
        }
        else{
            update(2*id+1,pos);
        }
        tree[id].no = tree[2*id].no +
tree[2*id+1].no;
};

int query(int id ,int left ,int right){
        if(tree[id].lson==left &&
tree[id].rson==right){
            return tree[id].no;
        }
        int mid = (tree[id].lson + tree[id].rson) /
2;
        if(right<=mid){
            return query(2*id,left,right);
        }
        else if(left > mid){
            return query(2*id+1,left,right);
        }
        else{
            return query(2*id,left,mid) +
query(2*id+1,mid+1,right);
        }
};
int main(){
        int n;
        long long ans ,min;
        while(~scanf("%d",&n)){
            build_tree(1,0,n-1);
            ans = 0;
            for(int i = 0;i<n;i++){
                scanf("%d",&num[i]);
                ans += query(1,num[i],n-1);
                update(1,num[i]);
            }
            min = ans;
            for(int i = 0;i<n-1;i++){
                ans = ans + (n - 2 * num[i] - 1);
                if(ans < min){
                    min = ans;
                }
            }
            printf("%lld\n",min);
        }
        return 0;
}
```

线段树（高级）

```
#include<cstdio>
#include<cstring>
#include<iostream>
#include<algorithm>
#define MAX 1000010
using namespace std;
struct Tree{
        int left ,right;
        __int64 sum;
        __int64 lnc;
}tree[MAX];
__int64 num[100010];
void build_tree(int id,int l,int r){
        tree[id].left = l;
        tree[id].right = r;
        tree[id].lnc = 0;
        if (l==r){
            tree[id].sum = num[l];
            return;
        }
        int mid = (l + r)>>1;
        build_tree(id<<1,l,mid);
        build_tree(id<<1+1,mid+1,r);
        tree[id].sum = tree[id<<1].sum +
tree[id<<1+1].sum;
};
void update(int id,int l,int r, __int64 val){
        if (tree[id].left==l&&tree[id].right==r){
            tree[id].lnc += val;
            return;
        }
```

```
        tree[id].sum += ((r - l + 1) * val);
        int mid = (tree[id].left + tree[id].right)>>1;
        if (r<=mid){
            update(id<<1,l,r,val);
        }
        else if(l>mid){
            update(id<<1+1,l,r,val);
        }
        else{
            update(id<<1,l,mid,val);
            update(id<<1+1,mid+1,r,val);
        }
};
__int64 query(int id,int l,int r){
        if (tree[id].left==l&&tree[id].right==r){
            return tree[id].sum + (r - l + 1) * tree[id].lnc;//询问总和
        }
        else{
            tree[id<<1].lnc += tree[id].lnc;
            tree[id<<1+1].lnc += tree[id].lnc;
            tree[id].sum += (tree[id].lnc * (tree[id].right - tree[id].left + 1));
            tree[id].lnc = 0;
        }
        int mid = (tree[id].left+tree[id].right)>>1;
        if (r<=mid){
            return query(id<<1,l,r);
        }
        else if(l>mid){
            return query(id<<1+1,l,r);
        }
        else{
            return query(id<<1,l,mid) + query(id<<1+1,mid+1,r);
        }
};
int main(){
        int n ,m ,q;
        while(scanf("%d%d",&n,&m)!=EOF){// 有 n 个数，进行 m 次操作
            memset(num,0,sizeof(num));
            memset(tree,0,sizeof(tree));
            for(int i = 1;i<=n;i++){
                scanf("%I64d",&num[i]);
            }
            build_tree(1,1,n);
            for(int i = 0;i<m;i++){
                char c;
                getchar();
                scanf("%c",&c);
                if(c=='C'){//C 表示增加
                    int l ,r;
                    __int64 val;

        scanf("%d %d %I64d",&l,&r,&val);
                    update(1,l,r,val);
                }
                if(c=='Q'){//Q 表示询问
                    int l ,r;
                    scanf("%d%d",&l,&r);

        printf("%I64d\n",query(1,l,r));//询问 l 到 r 区间内的总和
                }
            }
        }
        return 0;
}
```

二维树状数组

```
#include<cstdio>
#include<cstring>
#include<iostream>
#include<algorithm>
#define MAX 100005
using namespace std;
int N=1005,c[1005][1005];
int lowbit( int x ){
    return x & (-x);
}
void modify( int x, int y, int delta ){//a[x][y]增加 delta
    int i, j;
```

```
        for(i=x; i<=N; i+=lowbit(i)){
            for(j=y; j<=N; j+=lowbit(j)){
                c[i][j] += delta;
            }
        }
}
int sum( int x, int y ){
        int res = 0, i, j;
        for(i=x; i>0; i-=lowbit(i)){
            for(j=y; j>0; j-=lowbit(j)){
                res += c[i][j];
            }
        }
        return res;
}
```

离散化线段树扫描线

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cstdlib>
#include<cmath>
#include<map>
#include<set>
using namespace std;

__int64 ls[80005];//离散化
struct thanks{
        __int64 y,x1,x2;
        int flag;
}line[80005];//线段
int number,lsnum;

struct Tree{
        int left,right,flag;
        __int64 sum;
}tree[320005];
void build_tree(int id,int l,int r){
        tree[id].left = l;
        tree[id].right = r;
        if (l==r){
            tree[id].sum = 0;
            tree[id].flag=0;
            return;
        }
        int mid = (l + r)>>1;
        build_tree( (id<<1),l,mid);
        build_tree( (id<<1)+1,mid+1,r);
        tree[id].sum =0;
        tree[id].flag=0;
};
void update(int id,int l,int r, int val){
        if (tree[id].left==l && tree[id].right==r){
            tree[id].flag += val;

            if(tree[id].flag)
tree[id].sum=ls[r+1]-ls[l];
            else if(l==r) tree[id].sum=0;
            else
tree[id].sum=tree[id<<1].sum+tree[(id<<1)+1]
.sum;

            return;
        }
        int mid = (tree[id].left +
tree[id].right)>>1;
        if (r<=mid){
            update(id<<1,l,r,val);
        }
        else if(l>mid){
            update( (id<<1)+1,l,r,val);
        }
        else{
            update( (id<<1),l,mid,val);
            update( (id<<1)+1,mid+1,r,val);
        }
        if(tree[id].flag)
tree[id].sum=ls[tree[id].right+1]-ls[tree[id].lef
t];
        else
tree[id].sum=tree[id<<1].sum+tree[(id<<1)+1]
.sum;

};
bool cmp(struct thanks x,struct thanks y){
        return x.y<y.y;
}
```

```
void addline(__int64 y,__int64 x1,__int64 x2,__int64 flag){
    line[++number].y=y;
    line[number].x1=x1;
    line[number].x2=x2;
    line[number].flag=flag;
}
int main(){
    int i,j,m,n,T,vcase=0,last;
    __int64 x1,y1,x2,y2;
    while(scanf("%d",&n)!=EOF){
        number=0;lsnum=0;
        for(i=1;i<=n;i++){

    scanf("%I64d%I64d%I64d%I64d",&x1,&x2,&y1,&y2);
            ls[++lsnum]=x1;
            ls[++lsnum]=x2;

            if(x1<x2 && y1<y2){
                addline(y1,x1,x2,1);
                addline(y2,x1,x2,-1);
            }
        }
        sort(ls+1,ls+lsnum+1);
        last=1;
        for(i=2;i<=lsnum;i++){
            if(ls[i]!=ls[last]){
                ls[++last]=ls[i];
            }
        }
        lsnum=last;
        sort(line+1,line+number+1,cmp);
        build_tree(1,1,lsnum);
        __int64 sx,sy,ans=0;
        for(i=1;i<=number;i++){

    update(1,lower_bound(ls+1,ls+lsnum+1,line[i].x1)-ls,lower_bound(ls+1,ls+lsnum+1,line[i].x2)-ls-1,line[i].flag);
            sy=line[i+1].y-line[i].y;
            sx=tree[1].sum;
            ans+=sx*sy;
        }
        printf("%I64d\n",ans);
    }
    return 0;
}
```

数论

威佐夫博奕：ak =[k（1+ √ 5）/2]，bk= ak + k（k=0，1，2，…,n 方括号表示取整函数)

求和公式：

1 n*(n+1)/2

2 n*(n+1)*(2*n+1)/6

3 n*n*(n+1)*(n+1)/4

4 n*(n+1)*(6*n*n*n+9*n*n+n-1)/30

对于一个大于 1 正整数 n 可以分解质因数：n=p1^a1*p2^a2*p3^a3*…*pk^ak,

则 n 的正约数的个数就是（a1+1）（a2+1）（a3+1）…(ak+1) .

斯特林公式： n!=sqrt(2*3.14*n)*(n/e)^n

整数的唯一分解定理：任意正整数都有且只有一种方式写出其素因子的乘积表达式。

A=(p1^k1)*(p2^k2)*(p3^k3)*....*(pn^kn)

其中 pi 均为素数

约数和公式：对于已经分解的整数 A=(p1^k1)*(p2^k2)*(p3^k3)*....*(pn^kn)

有 A 的所有因子之和为：S = (1+p1+p1^2+p1^3+...p1^k1) * (1+p2+p2^2+p2^3+ … .p2^k2) * (1+p3+p3^3+ … +p3^k3) * .... * (1+pn+pn^2+pn^3+...pn^kn)（展开理解）

同余模公式：(a+b)%m=(a%m+b%m)%m

(a*b)%m=(a%m*b%m)%m

n 的环排列的个数与 n-1 个元素的排列的个数相等。

组合数公式：c[i][j]=c[i-1][j-1]+c[i-1][j];(c[i][0]=1,c[i][1]=i)

stirling 数（将 n 个人分成 k 个组，每组内按特定方式围圈的分组方式）公式：s[i][j]=s[i-1][j-1]+(i-1)*s[i-1][j];

（s[1][1]=1,s[i][0]=0）

M 个小球放入 N 个盒子中 C[N+M-1][N-1]

卡特兰数：

C[i+1]=(2*(2i+1)*C[i])/(i+2);

C[i]=for(j=0;j<=i-1;j++) C[i]+=C[j]*C[i-1-j];

A^B mod C=(A mod C)^(B mod Phi(C)) mod C

错排公式

```
void init()
{
    s[0]=0; s[1]=0; s[2]=1;
    int i;
   for (i=3;i<=100;i++)
       s[i]=(i-1)*(s[i-1]+s[i-2])%mod;
}
```

给定前 n 次游戏结果，计算第 n+1 次得分期望

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cstdlib>
#include<cmath>
using namespace std;
const int NM=6,NG=2,NS=4;//NM 表示一个骰子的面的个数,NG 表示之前进行的游戏次数,NS 表示骰子的个数
double p[2];//p[i]表示前面几次全部为 i 的条件下，下次为 1 的概率
double g[2];//g[i]表示前几次游戏中，i 对概率期望的贡献
double pv[15];//pv[i]表示每个骰子有 i 个面为红色的概率
int c[NM+1][NM+1];//组合数
double mianshu(int x){//计算每个骰子有 x 个面为 1 的概率
    return 1.0*c[NM][x]/pow(2.0,1.0*NM);
}
double chu(int x,int y){//计算当骰子有 x 个面为 1 的时候，连续掷出 y 的概率
    double vans=1.0;
    for(int i=1;i<=NG;i++) vans*=(y==1 ? 1.0*x/NM : 1.0*(NM-x)/NM);
    return vans;
}
double Pchu(int y){//计算骰子连续掷出 y 的总概率
    double vans=0.0;
    for(int i=0;i<=NM;i++) vans+=chu(i,y)*mianshu(i);
    return vans;
}
double getp(int y){//求解 p[y]
    double vans=0.0;
    for(int i=0;i<=NM;i++){

    vans+=1.0*i/NM*chu(i,y)*mianshu(i)/Pchu(y);
    }
    return vans;
}
void getC(){//得到组合数
    c[1][0]=1;
    c[1][1]=1;
    for(int i=2;i<=NM;i++){
        c[i][0]=1;
        for(int j=1;j<=i;j++){
            c[i][j]=c[i-1][j-1]+c[i-1][j];
        }
    }
}
int main(){
    int i,j,m,n,T;
    double ans;
    getC();
    p[0]=getp(0);
    p[1]=getp(1);

    g[0]=p[0]/NG;
    g[1]=p[1]/NG;
    scanf("%d",&T);
    while(T--){
        scanf("%d%d",&n,&m);
        ans=g[1]*(n+m)+g[0]*(NS*NG-n-m);
        printf("%.3lf\n",ans);
    }
    return 0;
```

```
}
```

中国剩余定理

对于 x=a[i] mod m[i];(1<=i<=r)
x=a[i]*M[i]*y[i] mod M;
M[i]=M/m[i]　　y[i]*M[i]=1 mod m[i];

高精度计算

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<algorithm>
using namespace std;
char a[1005],b[1005],c[1005];
int maxzhi(int n,int m){
    if(n>m) return n;
    else return m;
}
void jiafa(char *a,char *b){
    int i,n=strlen(a),m=strlen(b);
    for(i=0;i<n;i++){
        if(m-i-1>=0){
            c[i]+=a[n-i-1]+b[m-i-1]-48;
        }
        else{
            c[i]+=a[n-i-1];
        }
        if(c[i]>57){
            c[i]-=10;
            c[i+1]+=1;
            if(i==n-1){
                c[i+1]='1';
            }
        }
    }
    return;
}
void jianfa(char *a,char *b){
    int i,n=strlen(a),m=strlen(b);
    for(i=0;i<n;i++){
        if(m-i-1>=0){
            c[i]+=a[n-i-1]-b[m-i-1]+48;
        }
        else{
            c[i]+=a[n-i-1];
        }
        if(c[i]<48){
            c[i]+=10;
            c[i+1]-=1;
        }
    }
    return;
}
int main(){
    int i,j,m,n,afu,bfu,k,da,flag;
    while(scanf("%s %s",a,b)!=EOF){
        da=0;
        memset(c,0,sizeof(c));
        n=strlen(a);
        m=strlen(b);
        afu=0;
        bfu=0;
        if(a[0]=='-'){
            afu=1;
            for(i=0;i<n;i++)
                a[i]=a[i+1];
            n--;
        }
        if(b[0]=='-'){
            bfu=1;
            for(i=0;i<m;i++)
                b[i]=b[i+1];
            m--;
        }
        if(n>m        ||(n==m        &&
strcmp(a,b)>=0)){
            da=1;
            if(strcmp(a,b)==0 && afu==1)
                da=2;
            if(afu==bfu)
                jiafa(a,b);
            else
                jianfa(a,b);
        }
        else{
```

```
                da=2;
                if(afu==bfu)
                    jiafa(b,a);
                else
                    jianfa(b,a);
            }
            k=strlen(c);
            if( (afu==1 && da==1)||(bfu==1 && da==2) )
                printf("-");
            flag=0;
            for(i=k-1;i>=0;i--){
                if( flag==1||c[i]!='0'||k==1 ){
                    printf("%c",c[i]);
                    flag=1;
                }
            }
            printf("\n");
        }
        return 0;
}
```

快速幂

```
long long quickpow(long long m,long long n,long long k){
        long long b = 1;
        while (n > 0){
                if (n & 1)
                    b =( (b%k)*(m%k) )%k;
                n = n >> 1 ;
                m = ( (m%k)*(m%k) )%k;
        }
        return b;
}
```

线性筛法筛素数

```
void getprime(int n){
        int i,j;
        memset(flag,0,sizeof(flag));
        for(i=2;i<=n;i++){
                if(flag[i]==0) prime[++count1]=i;
                for(j=1;j<=count1 && i*prime[j]<=n;j++){
                        flag[i*prime[j]]=1;
                        if(i%prime[j]==0) break;
                }
        }
}
```

快速 GCD

```
int gcd(int a, int b)
{
    while ( b ^= a ^= b ^= a %= b );
    return a;
}
```

欧拉值（1 到 n-1 与 n 互质的个数）

```
unsigned euler(unsigned x){// 就是公式
    unsigned i, res=x;
    for (i = 2; i < (int)sqrt(x * 1.0) + 1; i++)
        if(x%i==0) {
            res = res / i * (i - 1);
            while (x % i == 0) x /= i; //保证i一定是素数
        }
        if (x > 1) res = res / x * (x - 1);
        return res;
}
```

扩展欧几里得函数

```
int extendeGcd(int a,int b){
    if(b==0){
        x=1;
        y=0;
        return a;
    }
    else{
        int tmp=extendeGcd(b,a%b);
        int t=x;
        x=y;
        y=t-a/b*y;
        return tmp;
```

```
    }
}
```

对于 a*x+b*y=d,一定存在满足 d=gcd(a,b)的解(x,y),返回求解。

贝祖定理：对于 a*x+b*y=c 来说，c 一定是 gec(a,b)的整数倍。

高斯消元求解方程

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cstdlib>
#include<cmath>
#include<iostream>
using namespace std;

const int maxn = 105;

int equ, var; // 有 equ 个方程，var 个变元。增广阵行数为 equ, 分别为 0 到 equ - 1，列数为 var + 1，分别为 0 到 var.
int a[maxn][maxn];
int x[maxn]; // 解集.
bool free_x[maxn]; // 判断是否是不确定的变元.
int free_num;

void Debug(void){
    int i, j;
    for (i = 0; i < equ; i++){
        for (j = 0; j < var + 1; j++){
            cout << a[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

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

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

// 高斯消元法解方程组(Gauss-Jordan elimination).(-2 表示有浮点数解，但无整数解，-1 表示无解，0 表示唯一解，大于 0 表示无穷解，并返回自由变元的个数)
int Gauss(void){
    int i, j, k;
    int max_r; // 当前这列绝对值最大的行.
    int col; // 当前处理的列.
    int ta, tb;
    int LCM;
    int temp;
    int free_x_num;
    int free_index;
    // 转换为阶梯阵.
    col = 0; // 当前处理的列.
    for (k = 0; k < equ && col < var; k++, col++){ // 枚举当前处理的行.
        // 找到该 col 列元素绝对值最大的那行与第 k 行交换.(为了在除法时减小误差)
        max_r = k;
        for (i = k + 1; i < equ; i++){
            if (abs(a[i][col]) > abs(a[max_r][col])) max_r = i;
        }
        if (max_r != k){ // 与第 k 行交换.
            for (j = k; j < var + 1; j++) swap(a[k][j], a[max_r][j]);
        }
        if (a[k][col] == 0){ // 说明该 col 列第 k 行以下全是 0 了，则处理当前行的下一列.
            k--; continue;
        }
        for (i = k + 1; i < equ; i++){ // 枚举
```

要删去的行.

```
                if (a[i][col] != 0){
                    LCM = lcm(abs(a[i][col]), abs(a[k][col]));
                    ta = LCM / abs(a[i][col]), tb = LCM / abs(a[k][col]);
                    if (a[i][col] * a[k][col] < 0) tb = -tb; // 异号的情况是两个数相加.
                    for (j = col; j < var + 1; j++){
                        a[i][j] = a[i][j] * ta - a[k][j] * tb;
                    }
                }
            }
        }
        // 1. 无解的情况: 化简的增广阵中存在(0, 0, ..., a)这样的行(a != 0).
        for (i = k; i < equ; i++){ // 对于无穷解来说，如果要判断哪些是自由变元，那么初等行变换中的交换就会影响，则要记录交换.
            if (a[i][col] != 0) return -1;
        }
        // 2. 无穷解的情况: 在 var * (var + 1)的增广阵中出现(0, 0, ..., 0)这样的行，即说明没有形成严格的上三角阵.
        // 且出现的行数即为自由变元的个数.
        if (k < var){
            // 首先，自由变元有 var - k 个，即不确定的变元至少有 var - k 个.
            for (i = k - 1; i >= 0; i--){
                // 第 i 行一定不会是(0, 0, ..., 0)的情况，因为这样的行是在第 k 行到第 equ 行.
                // 同样，第 i 行一定不会是(0, 0, ..., a), a != 0 的情况，这样的无解的.
                free_x_num = 0; // 用于判断该行中的不确定的变元的个数，如果超过 1 个，则无法求解，它们仍然为不确定的变元.
                for (j = 0; j < var; j++){
                    if (a[i][j] != 0 && free_x[j]) free_x_num++, free_index = j;
                }
                if (free_x_num > 1) continue; // 无法求解出确定的变元.
                // 说明就只有一个不确定的变元 free_index，那么可以求解出该变元，且该变元是确定的.
                temp = a[i][var];
                for (j = 0; j < var; j++){
                    if (a[i][j] != 0 && j != free_index) temp -= a[i][j] * x[j];
                }
                x[free_index] = temp / a[i][free_index]; // 求出该变元.
                free_x[free_index] = 0; // 该变元是确定的.
            }
            return var - k; // 自由变元有 var - k 个.
        }
        // 3. 唯一解的情况: 在 var * (var + 1)的增广阵中形成严格的上三角阵.
        // 计算出 Xn-1, Xn-2 ... X0.
        for (i = var - 1; i >= 0; i--){
            temp = a[i][var];
            for (j = i + 1; j < var; j++){
                if (a[i][j] != 0) temp -= a[i][j] * x[j];
            }
            if (temp % a[i][i] != 0) return -2; // 说明有浮点数解，但无整数解.
            x[i] = temp / a[i][i];
        }
        return 0;
    }

    int main(void){
        int i, j;
        while (scanf("%d %d", &equ, &var) != EOF){
            memset(a, 0, sizeof(a));
            memset(x, 0, sizeof(x));
            memset(free_x, 1, sizeof(free_x)); // 一开始全是不确定的变元.
            for (i = 0; i < equ; i++){
                for (j = 0; j < var + 1; j++){
                    scanf("%d", &a[i][j]);
```

```
            }
        }
        //Debug();
        free_num = Gauss();
        if (free_num == -1) printf("无解!\n");
        else if (free_num == -2) printf("有浮点数解，无整数解!\n");
        else if (free_num > 0){
            printf("无穷多解! 自由变元个数为%d\n", free_num);
            for (i = 0; i < var; i++){
                if (free_x[i]) printf("x%d是不确定的\n", i + 1);
                else printf("x%d: %d\n", i + 1, x[i]);
            }
        }
        else{
            for (i = 0; i < var; i++){
                printf("x%d: %d\n", i + 1, x[i]);
            }
        }
        printf("\n");
    }
    return 0;
}
```

高斯消元求解开关灯游戏

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cstdlib>
#include<cmath>
#include<iostream>
using namespace std;

const int maxn = 305;

int equ, var; // 有 equ 个方程，var 个变元。增广阵行数为 equ, 分别为 0 到 equ - 1，列数为 var + 1，分别为 0 到 var.
int a[maxn][maxn];
int x[maxn]; // 解集.
bool free_x[maxn]; // 判断是否是不确定的变元.
int free_num;

void Debug(void){
    int i, j;
    for (i = 0; i < equ; i++){
        for (j = 0; j < var + 1; j++){
            cout << a[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

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

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

// 高斯消元法解方程组(Gauss-Jordan elimination).(-2 表示有浮点数解，但无整数解，-1 表示无解，0 表示唯一解，大于 0 表示无穷解，并返回自由变元的个数)
int Gauss(void){
    int i, j, k;
    int max_r; // 当前这列绝对值最大的行.
    int col; // 当前处理的列.
    int ta, tb;
    int LCM;
    int temp;
    int free_x_num;
```

```
    int free_index;
    // 转换为阶梯阵.
    col = 0; // 当前处理的列.
    for (k = 0; k < equ && col < var; k++, col++){ // 枚举当前处理的行.
        // 找到该 col 列元素绝对值最大的那行与第 k 行交换.(为了在除法时减小误差)
        max_r = k;
        for (i = k + 1; i < equ; i++){
            if (abs(a[i][col]) > abs(a[max_r][col])) max_r = i;
        }
        if (max_r != k){ // 与第 k 行交换.
            for (j = k; j < var + 1; j++) swap(a[k][j], a[max_r][j]);
        }
        if (a[k][col] == 0){ // 说明该 col 列第 k 行以下全是 0 了，则处理当前行的下一列.
            k--; continue;
        }
        for (i = k + 1; i < equ; i++){ // 枚举要删去的行.
            if (a[i][col] != 0){
                LCM = lcm(abs(a[i][col]), abs(a[k][col]));
                for (j = col; j < var + 1; j++){
                    a[i][j] = a[i][j]^a[k][j];
                }
            }
        }
    }
    // 1. 无解的情况: 化简的增广阵中存在(0, 0, ..., a)这样的行(a != 0).
    for (i = k; i < equ; i++){ // 对于无穷解来说，如果要判断哪些是自由变元，那么初等行变换中的交换就会影响，则要记录交换.
        if (a[i][col] != 0) return -1;
    }
    // 2. 无穷解的情况: 在 var * (var + 1)的增广阵中出现(0, 0, ..., 0)这样的行，即说明没有形成严格的上三角阵.
    // 且出现的行数即为自由变元的个数.
    if (k < var){
        for (i = k - 1; i >= 0; i--){
            // 第 i 行一定不会是(0, 0, ..., 0)的情况，因为这样的行是在第 k 行到第 equ 行.
            // 同样，第 i 行一定不会是(0, 0, ..., a), a != 0 的情况，这样的无解的.
            free_x_num = 0; // 用于判断该行中的不确定的变元的个数，如果超过 1 个，则无法求解，它们仍然为不确定的变元.
            for (j = 0; j < var; j++){
                if (a[i][j] != 0 && free_x[j]) free_x_num++, free_index = j;
            }
            if (free_x_num > 1) continue; // 无法求解出确定的变元.
            // 说明就只有一个不确定的变元 free_index，那么可以求解出该变元，且该变元是确定的.
            temp = a[i][var];
            for (j = 0; j < var; j++){
                if (a[i][j] != 0 && j != free_index) temp ^= a[i][j] * x[j];
            }
            x[free_index] = temp / a[i][free_index]; // 求出该变元.
            free_x[free_index] = 0; // 该变元是确定的.
        }
        return var - k; // 自由变元有 var - k 个.
    }
    // 3. 唯一解的情况: 在 var * (var + 1)的增广阵中形成严格的上三角阵.
    // 计算出 Xn-1, Xn-2 ... X0.
    for (i = var - 1; i >= 0; i--){
        temp = a[i][var];
        for (j = i + 1; j < var; j++){
            if (a[i][j] != 0) temp ^= a[i][j] * x[j];
        }
        if (temp % a[i][i] != 0) return -2; // 说明有浮点数解，但无整数解.
        x[i] = temp / a[i][i];
```

```
    }
    return 0;
}

int map[20][20],dir[4][2]={-1,0,0,1,1,0,0,-1};
int main(void){
    int i,j,T,n,m,vcase=0;
    char c;
    scanf("%d",&T);
    while(T--){
        scanf("%d",&n);
        getchar();
        m=n;
        for(i=1;i<=n;i++){
            for(j=1;j<=m;j++){
                scanf("%c",&c);
                if(c=='y') map[i][j]=0;
                else map[i][j]=1;
            }
            getchar();
        }
        memset(a, 0, sizeof(a));
        memset(x, 0, sizeof(x));
        memset(free_x, 1, sizeof(free_x)); //一开始全是不确定的变元.
        equ=n*n;
        var=n*n;
        for(i=0;i<n;i++){
            for(j=0;j<m;j++){
                int hang=i*m+j;
                for(int l=0;l<4;l++){
                    int nowx=i+dir[l][0];
                    int nowy=j+dir[l][1];
                    if(nowx>=0 && nowx<n && nowy>=0 && nowy<m){
                        int e=nowx*m+nowy;
                        a[hang][e]=1;
                    }
                }
                a[hang][hang]=1;

    a[hang][var]=map[i+1][j+1];
            }
        }
        //Debug();
        free_num = Gauss();
        if(free_num==0){
            int ans=0;
            for(i=0;i<n;i++){
                for(j=0;j<m;j++){
                    int p=i*m+j;
                    ans+=x[p];
                }
            }
            printf("%d\n",ans);
        }
        else if(free_num==-1){
            printf("inf\n");
        }
        else{
            int ans=999999999;
            int vz=(1<<free_num)-1;
            int k=var-free_num;
            for(int st=0;st<vz;st++){
                int free_x_num,free_index,temp;
                memset(free_x, 1, sizeof(free_x));
                for(i=var-1;i>=k;i--){
                    j=i-k+1;

    x[i]=( ( (1<<(j-1) )&st)==0 ? 0:1);
                    free_x[i]=0;
                }

                for (i = k - 1; i >= 0; i--){
                    free_x_num = 0;
                    for (j = 0; j < var; j++){
                        if (a[i][j] != 0 && free_x[j]) free_x_num++, free_index = j;
                    }
                    if (free_x_num > 1) continue;
                    temp = a[i][var];
                    for (j = 0; j < var; j++){
                        if (a[i][j] != 0 && j != free_index) temp ^= a[i][j] * x[j];
```

```
                        }
                        x[free_index] = temp / a[i][free_index];
                        free_x[free_index] = 0;
                    }
                    int vans=0;
                    for(i=0;i<n;i++){
                        for(j=0;j<m;j++){
                            int p=i*m+j;
                            vans+=x[p];
                        }
                    }
                    ans=min(ans,vans);
                }
                printf("%d\n",ans);
            }
        }
        return 0;
}
```

求斐波那契数列对 n 的循环节

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

using namespace std;
typedef unsigned long long LL;

const int M = 2;

struct Matrix{
    LL m[M][M];
};

Matrix A;
Matrix I = {1,0,0,1};

Matrix multi(Matrix a,Matrix b,LL MOD){//矩阵乘法
    Matrix c;
    for(int i=0; i<M; i++){
        for(int j=0; j<M; j++){
            c.m[i][j] = 0;
            for(int k=0; k<M; k++)
                c.m[i][j] = (c.m[i][j]%MOD + (a.m[i][k]%MOD)*(b.m[k][j]%MOD)%MOD)%MOD;
            c.m[i][j] %= MOD;
        }
    }
    return c;
}

Matrix power(Matrix a,LL k,LL MOD){//矩阵快速幂 a^k
    Matrix ans = I,p = a;
    while(k){
        if(k & 1){
            ans = multi(ans,p,MOD);
            k--;
        }
        k >>= 1;
        p = multi(p,p,MOD);
    }
    return ans;
}

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

const int N = 400005;//质数的范围大小
const int NN = 5005;//n 的质因子数

LL num[NN],pri[NN];//pri 存放能被 n 整除的质数，num 存放对应的幂数.对于 n=12，pri[0]=2,num[0]=2.  2^2=4,4|12
LL fac[NN];//存放因子
int cnt,c;
```

```
bool prime[N];//判断 n 是否为质数,是的话 prime[n]为 1
int p[N];//存放质数，p[i]表示第 i 个质数，从 0 开始， p[0]=2
int k;

void isprime(){//标记并记录 0-N 的质数
    k = 0;
    memset(prime,true,sizeof(prime));
    for(int i=2; i<N; i++){
        if(prime[i]){
            p[k++] = i;
            for(int j=i+i; j<N; j+=i)
                prime[j] = false;
        }
    }
}

LL quick_mod(LL a,LL b,LL m){//快速幂，a^b%m 的值
    LL ans = 1;
    a %= m;
    while(b){
        if(b & 1){
            ans = ans * a % m;
            b--;
        }
        b >>= 1;
        a = a * a % m;
    }
    return ans;
}

LL legendre(LL a,LL p){//判断 a 是否是 p 的二次剩余，应用了欧拉给出的判别条件
    if(quick_mod(a,(p-1)>>1,p)==1) return 1;
    else
return -1;
}

void Solve(LL n,LL pri[],LL num[]){//分解 n，求解出质因子的个数和相应的幂数
    cnt = 0;
    LL t = (LL)sqrt(1.0*n);
    for(int i=0; p[i]<=t; i++){
        if(n%p[i]==0){
            int a = 0;
            pri[cnt] = p[i];
            while(n%p[i]==0){
                a++;
                n /= p[i];
            }
            num[cnt] = a;
            cnt++;
        }
    }
    if(n > 1){
        pri[cnt] = n;
        num[cnt] = 1;
        cnt++;
    }
}

void Work(LL n){//统计 n 的因子个数，放入 fac 数组中
    c = 0;
    LL t = (LL)sqrt(1.0*n);
    for(int i=1; i<=t; i++){
        if(n % i == 0){
            if(i * i == n) fac[c++] = i;
            else{
                fac[c++] = i;
                fac[c++] = n / i;
            }
        }
    }
}

LL find_loop(LL n){
    Solve(n,pri,num);
    LL ans=1;
    for(int i=0; i<cnt; i++){
        LL record=1;
        if(pri[i]==2)
            record=3;
        else if(pri[i]==3)
            record=8;
        else if(pri[i]==5)
```

```
                record=20;
            else{
                if(legendre(5,pri[i])==1)
                    Work(pri[i]-1);
                else
                    Work(2*(pri[i]+1));
                sort(fac,fac+c);
                for(int k=0; k<c; k++){//在 c 个因子中寻找最小循环节
                    Matrix a = power(A,fac[k]-1,pri[i]);
                    LL x = (a.m[0][0]%pri[i]+a.m[0][1]%pri[i])%pri[i];
                    LL y = (a.m[1][0]%pri[i]+a.m[1][1]%pri[i])%pri[i];
                    if(x==1 && y==0){
                        record = fac[k];
                        break;
                    }
                }
            }
            for(int k=1; k<num[i]; k++)//*p^(m-1)
                record *= pri[i];
            ans = ans/gcd(ans,record)*record;
        }
        return ans;
}

void Init(){
        A.m[0][0] = 1;
        A.m[0][1] = 1;
        A.m[1][0] = 1;
        A.m[1][1] = 0;
}

int main(){
        LL n;
        Init();
        isprime();
        while(cin>>n)
            cout<<find_loop(n)<<endl;
        return 0;
}
```

运算大组合数对质数取模

```
typedef __int64 ll;
const int maxn = 100005;
const ll mod = 1000000009;
ll factorial[2*maxn];

void init(){
        factorial[0] = 1;
        for(int i = 1;i< 2*maxn;i++){
            factorial[i] = (factorial[i-1]*i)%mod;
        }
}

ll powmod(ll a, ll m, ll n){
        ll ret = 1;
        a %= n;
        while(m){
            if(m&1)
ret=ret*a%n;a=a*a%n;m>>=1;
        }
        return ret;
}

ll calCmn(ll p, ll q){
        if(q==1) return p;
        ll x=q, y=p-q;
        ll tmp=(factorial[p]%mod) * powmod(factorial[y]*factorial[x], mod-2, mod)%mod;
        return tmp;
}
```

运算组合数对所有数取模

```
typedef __int64 ll;
const int maxn = 100005;
const ll mod = 1000000009;
ll factorial[maxn];

void init() {
        factorial[0] = 1;
        for(int i = 1; i< maxn; i++)
```

```
        factorial[i] = (factorial[i-1] * i) % mod;
}
ll exgcd( ll a, ll b, ll &x, ll &y) {
    if(!b) {
        x=1, y=0;
        return a;
    }
    ll ret = exgcd(b, a%b, x, y);
    ll t=x, x=y, y=t-a/b*y;
    return ret;
}
ll InverseMod(ll a, ll n) {
    if( n<= 0)return -1;
    ll x, y;
    ll comDiv = exgcd( a, n, x, y);
    if( comDiv != -1) return -1;
    return ((x%n)+n)%n;
}
ll calCmn(ll p, ll q) {
    ll x = q, y = p - q;
    ll tmp = (factorial[p] % mod) * InverseMod(factorial[y]*factorial[x], mod) % mod;
    return tmp;
}
```

动态规划

四边形优化 dp

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cstdlib>
#include<cmath>
using namespace std;
struct thanks{
    int x,y;
}p[1005];
int dp[1005][1005],s[1005][1005];
int main(){
    int i,j,m,n,l,k;
    while(scanf("%d",&n)!=EOF){
        for(i=1;i<=n;i++){
            scanf("%d%d",&p[i].x,&p[i].y);
        }
        memset(dp,-1,sizeof(dp));
        for(l=1;l<=n;l++){
            for(i=1;i+l-1<=n;i++){
                j=i+l-1;
                if(i==j){
                    dp[i][j]=0;
                    s[i][j]=i;
                    continue;
                }
                for(k=s[i][j-1];k<=s[i+1][j] && k<j;k++){
                    if(dp[i][j]==-1 || dp[i][j]>dp[i][k]+dp[k+1][j]+p[k+1].x-p[i].x+p[k].y-p[j].y){
                        s[i][j]=k;
    dp[i][j]=dp[i][k]+dp[k+1][j]+p[k+1].x-p[i].x+p[k].y-p[j].y;
                    }
                }
            }
        }
        printf("%d\n",dp[1][n]);
    }
    return 0;
}
```

s[i][j]表示区间 i，j 的最优解中点下标。
dp[i][j]的最优解更新点在 s[i][j-1],s[i+1][j]之间。
dp[i,j]=min(dp[i,k]+dp[k+1,j]+cost(i,j))
cost(i,j)需要满足以下条件方可使用西边形 dp：
cost(a,c)+cost(b,d)<=cost(b,c)+cost(a,d)
cost(b,c)<=cost(a,d)

斜率优化 dp

```
#include<cstdio>
#include<cstring>
```

```
#include<algorithm>
#include<cstdlib>
#include<cmath>
#include<queue>
using namespace std;
__int64 a[500005],sum[500005],dp[500005];
int qst,qed,q[500005];
__int64 X(int x){//计算 X 值,x 为数组下标
    return 2*sum[x];
}
__int64 Y(int x){//计算 Y 值，x 为数组下标
    return dp[x]+sum[x]*sum[x];
}
bool compare_K(int pa,int pb,int pc){//比较斜率与斜率的大小
    return
(Y(pc)-Y(pb))*(X(pb)-X(pa))<=(Y(pb)-Y(pa))*(X(pc)-X(pb));
}
bool compare_K_value(int pa,int pb,__int64 value){//比较斜率与值的大小
    return Y(pb)-Y(pa)<=(X(pb)-X(pa))*value;
}
int main(){
    int i,j,m,n;
    while(scanf("%d%d",&n,&m)!=EOF){
        memset(sum,0,sizeof(sum));
        memset(dp,0,sizeof(dp));
        for(i=1;i<=n;i++){
            scanf("%I64d",&a[i]);
            sum[i]=sum[i-1]+a[i];
        }
        qed=0;
        qst=1;
        q[++qed]=0;
        for(i=1;i<=n;i++){
            while( qed-qst+1>=2 &&
compare_K_value(q[qst],q[qst+1],sum[i]) ){//
从队头寻找符合条件的点
                qst++;
            }

    dp[i]=dp[q[qst]]+m+(sum[i]-sum[q[qst]])
*(sum[i]-sum[q[qst]]);//更新 DP 值
            while( qed-qst+1>=2 &&
compare_K(q[qed-1],q[qed],i) ){//寻找插入的
位置
                qed--;
            }
            q[++qed]=i;
        }
        printf("%I64d\n",dp[n]);
    }
    return 0;
}
```

dp[i]=dp[j]+sum[i]-sum[j];
dp[j]+(sum[i]-sum[j])^2+m>dp[k]+(sum[i]-sum[k])^2+m
((dp[j]+sum[j]^2)-(dp[k]+sum[k]^2))/(2*sum[j]-2*sum[k])>sum[i]
Y(j)=dp[j]+sum[j]^2;
X(j)=2*sum[j];
又因为 sum[i]递增，所以如果斜率(j,i)>sum[i],则 j 点可删。
且队列中两点斜率必须递增。

字符串

KMP

KMP 第二种表示方法 NEXT[i]表示第 i 个失配后从 next[i]开始匹配，next[i]为下标。

```
void getnext(char s[],int pre[]){
    int i=0, j=-1,n=(int)strlen(s);
    pre[0]=-1;
    while(i<=n){
        if(j==-1 || s[j]==s[i]){
            i++; j++;
            pre[i]=j;
        }
        else j=pre[j];
    }
}
```

特殊性质：

1.找一个字符串前 i 个和后 i 个一样的话（正序），只需要顺着 Next[n]找即可找到所有的 i。

2.找一个字符串最多可由几个子串构成，需要用总长度-Next[n]得出最小公共元，然后顺着 Next[n]去整除公共元，全部可以整除，即说明这一段就是最小的公共元。

匹配函数：

```
int my_KMP(char *S, char *T, int pos) {
int i = pos, j = 0;//pos(S 的下标 0≤pos<StrLength(S))
while ( S[i] != '\0' && T[j] != '\0' ) {
      if (S[i] == T[j] ) {
            ++i;
            ++j;
       }
    else{
        i++;
       j = next[j];
    }
}
if (T[j] == '\0')
      return (i-j);
else
       return -1;
}

int kmpCount(char *da,char* xiao,int tlen,int plen)//第一个字符串去组成第二个字符串所需要第一个字符串的个数
{
      int i=0;
      int j=0;
      int result=0;
      while(i<=tlen-1)
      {
            if(j==-1||da[i]==xiao[j])
            {
                  i++;
                  j++;
            }
            else
                  j=mnext[j];
            if(j==plen)
                  result++;
      }
      return result;
}
```

KMP 第一种表示方法

next[i]表示在点 i 前面有 next[i]个字符和字符串首的 next[i]个字符相等，next[i]为数量

```
void get_nextval(const char *T, int next[])
{
      int j = 0, k = -1;
      next[0] = -1;
      while ( T[j/*+1*/] != '\0' )
      {
            if (k == -1 || T[j] == T[k])
            {
                  ++j; ++k;
                  if (T[j]!=T[k])
                        next[j] = k;
                  else
                        next[j] = next[k];
            }
            else
                  k = next[k];
      }
}


void getNext(const char* pattern,int next[])
{
      next[0]= -1;
      int k=-1,j=0;
      while(pattern[j] != '\0')
      {
            if(k!= -1 && pattern[k]!= pattern[j] )
                  k=next[k];
            ++j;++k;
            if(pattern[k]== pattern[j])
                  next[j]=next[k];
            else
```

```
                next[j]=k;
        }
}
```

匹配函数：

```
int KMP(const char *Text,const char* Pattern)
//const 表示函数内部不会改变这个参数的值。
{
        if(             !Text||!Pattern||Pattern[0]=='\0'  ||  Text[0]=='\0' )//
                return -1;//空指针或空串，返回-1。
        int len=0;
        const char * c=Pattern;
        while(*c++!='\0')//移动指针比移动下标快。
        {
                ++len;//字符串长度。
        }
        int *next=new int[len+1];
        get_nextval(Pattern,next);//      求 Pattern 的 next 函数值
        int index=0,i=0,j=0;
        while(Text[i]!='\0'             && Pattern[j]!='\0' )
        {
                if(Text[i]== Pattern[j])
                {
                        ++i;// 继续比较后继字符
                        ++j;
                }
                else
                {
                        index += j-next[j];
                        if(next[j]!=-1)
                                j=next[j];//模式串向右移动
                        else
                        {
                                j=0;
                                ++i;
                        }
                }
        }//while
        delete []next;
        if(Pattern[j]=='\0')
                return index;// 匹配成功
        else
                return -1;
}
```

后缀数组

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cstdlib>
using namespace std;
int wa[200005],wb[200005],wv[200005],ws[200005];
int cmp(int *r,int a,int b,int l){
    return r[a]==r[b]&&r[a+l]==r[b+l];
}
void da(int *r,int *sa,int n,int m){
    int i,j,p,*x=wa,*y=wb,*t;
    for(i=0;i<m;i++) ws[i]=0;
    for(i=0;i<n;i++) ws[x[i]=r[i]]++;
    for(i=1;i<m;i++) ws[i]+=ws[i-1];
    for(i=n-1;i>=0;i--) sa[--ws[x[i]]]=i;
    for(j=1,p=1;p<n;j*=2,m=p){
        for(p=0,i=n-j;i<n;i++) y[p++]=i;
        for(i=0;i<n;i++)            if(sa[i]>=j) y[p++]=sa[i]-j;
        for(i=0;i<n;i++) wv[i]=x[y[i]];
        for(i=0;i<m;i++) ws[i]=0;
        for(i=0;i<n;i++) ws[wv[i]]++;
        for(i=1;i<m;i++) ws[i]+=ws[i-1];
        for(i=n-1;i>=0;i--)
sa[--ws[wv[i]]]=y[i];

    for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)

    x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
```

```
    }
    return;
}
int rank1[200005],height[200005];
void calheight(int *r,int *sa,int n){
    int i,j,k=0;
    for(i=1;i<=n;i++) rank1[sa[i]]=i;
    for(i=0;i<n;height[rank1[i++]]=k)

    for(k?k--:0,j=sa[rank1[i]-1];r[i+k]==r[j+k];
k++);
    return;
}
int rmq[200005][20];
void initrmq(int n){
    int i, k;
    for(i = 2; i <= n; ++i)
        rmq[i][0] = height[i];
    for(k = 1; (1 << k) <= n; ++k){
        for(i = 2; i + (1 << k) - 1 <= n; ++i){
            rmq[i][k] = min(rmq[i][k -
1],rmq[i + (1 << (k - 1))][k - 1]);
        }
    }
}
int lcp(int a, int b){
    a = rank1[a], b = rank1[b];
    if(a > b)
        swap(a, b);
    ++a;
    int k = (int) (log((b - a + 1) * 1.0) /
log(2.0));
    return min(rmq[a][k], rmq[b - (1 << k) +
1][k]);
}
```

da(r,sa,n,m);（m 大于最大值，n 为长度，注意将 r[n-1]添加个 0）

```
calheight(r,sa,n-1);（求出 height 数组）
initrmq(n-1);
```

计算几何

1.求面积，用向量差乘 （a,b）X（c,d）=a*d-b*c，该算式会自动区分正负，化成若干个三角形面积相加即可。

2.求重心，实心多边形需要算出各个三角形的重心，然后乘面积算出重心坐标的比例，叠加除以总面积即可

3.求凸包，选取 y 值最小的一点（若有多个，选取 x 值小的），然后将其他点按逆时针排列，对每一个点按逆时针顺序进行判断，向量（n-2,n-1）X(n-1,n)，如果为负值，就删除。

多边形面积

```
#include<stdio.h>
#include<string.h>
#include<algorithm>
#include<math.h>
using namespace std;
struct point{
    double x;
    double y;
}map[100005];
int main(){
    int i,j,m,n;
    double x,y,x1,y1,x2,y2,ans,a,b,c,d;
    while(scanf("%d",&n)!=EOF){
        ans=0;
        for(i=1;i<=n;i++)

    scanf("%lf%lf",&map[i].x,&map[i].y);
        x=map[1].x;
        y=map[1].y;
        for(i=2;i<=n-1;i++){
            x1=map[i].x;
            y1=map[i].y;
            x2=map[i+1].x;
            y2=map[i+1].y;
            a=x-x1;
            b=y-y1;
            c=x-x2;
            d=y-y2;

    //ans+=(sqrt(a*a+b*b)*sqrt(c*c+d*d)*sq
rt(1-pow(  (a*c+b*d)/(sqrt(a*a+b*b)*sqrt(c*c
+d*d) ),2) ) )/2;
```

```
            ans+=(a*d-b*c)/2;
        }
        ans=fabs(ans);
        printf("%.2lf\n",ans);
    }
    return 0;
}
```

多边形重心

```
#include<stdio.h>
#include<string.h>
#include<algorithm>
using namespace std;
struct point{
    double x;
    double y;
}map[100005];
int main(){
    int i,j,m,n;
    double x,y,ss,sx,sy,s;
    while(scanf("%d",&n)!=EOF){
        ss=0;
        sx=0;
        sy=0;
        for(i=1;i<=n;i++)

    scanf("%lf%lf",&map[i].x,&map[i].y);
        for(i=2;i<=n-1;i++){

    x=(map[1].x+map[i].x+map[i+1].x)/3;

    y=(map[1].y+map[i].y+map[i+1].y)/3;

    s=( (map[i].x-map[1].x)*(map[i+1].y-map[
1].y)-(map[i+1].x-map[1].x)*(map[i].y-map[1].
y) )/2;
            sx+=s*x;
            sy+=s*y;
            ss+=s;
        }

    printf("%.2lf %.2lf\n",sx/ss+0.00000001,s
y/ss+0.00000001);
    }
    return 0;
}
```

分治法求最小点对（二维）

```
/*分治算法求最小点对*/
#include <iostream>
#include <cmath>
#include <algorithm>
#define MAXN 100005
using namespace std;
struct Point{
    double x,y;
};
struct Point
point[MAXN],*px[MAXN],*py[MAXN];
double get_dis(Point *p1,Point *p2){
    return
sqrt((p1->x-p2->x)*(p1->x-p2->x)+(p1->y-p2->
y)*(p1->y-p2->y));
}
bool cmpx(Point *p1,Point *p2) {return
p1->x<p2->x;}
bool cmpy(Point *p1,Point *p2) {return
p1->y<p2->y;}
double min(double a,double b){return
a<b?a:b;}
//-------核心代码----------//
double closest(int s,int e){
    if(s+1==e)
        return get_dis(px[s],px[e]);
    if(s+2==e)
        return
min(get_dis(px[s],px[s+1]),min(get_dis(px[s+1
],px[e]),get_dis(px[s],px[e])));
    int mid=(s+e)>>1;
    double
ans=min(closest(s,mid),closest(mid+1,e));// 递
归求解
    int i,j,cnt=0;
    for(i=s;i<=e;i++){// 把 x 坐 标 在
px[mid].x-ans~px[mid].x+ans 范围内的点取
出来
```

```
    if(px[i]->x>=px[mid]->x-ans&&px[i]->x<=px[mid]->x+ans)
                py[cnt++]=px[i];
    }
    sort(py,py+cnt,cmpy);//按 y 坐标排序
    for(i=0;i<cnt;i++){
        for(j=i+1;j<cnt;j++){//py 数组中的点是按照 y 坐标升序的
                if(py[j]->y-py[i]->y>=ans)
                    break;

    ans=min(ans,get_dis(py[i],py[j]));
        }
    }
    return ans;
}
int main(){
    int i,n;
    while(scanf("%d",&n)!=EOF){
        if(n==0)
            break;
        for(i=0;i<n;i++){

    scanf("%lf%lf",&point[i].x,&point[i].y);
            px[i]=&point[i];
        }
        sort(px,px+n,cmpx);
        double distance=closest(0,n-1);
        printf("%.2lf\n",distance);
    }
    return 0;
}
```

先按 X 坐标将点分成两部分，分别算出两部分的最小距离（递归），然后选出分界线左右的点，按 y 坐标从小到大排序，遍历里面对每两个点，更新距离。

分治法求最小点对（三维）

```
/*分治算法求最小点对*/
#include <iostream>
#include <cmath>
#include <algorithm>
#define MAXN 100005
using namespace std;
struct Point{
    double x,y,z;
};
struct Point point[MAXN],*px[MAXN],*py[MAXN];
double get_dis(Point *p1,Point *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));
}
bool cmpx(Point *p1,Point *p2) {return p1->x<p2->x;}
bool cmpy(Point *p1,Point *p2) {return p1->y<p2->y;}
double min(double a,double b){return a<b?a:b;}
//-------核心代码------------//
double closest(int s,int e){
    if(s+1==e)
        return get_dis(px[s],px[e]);
    if(s+2==e)
        return min(get_dis(px[s],px[s+1]),min(get_dis(px[s+1],px[e]),get_dis(px[s],px[e])));
    int mid=(s+e)>>1;
    double ans=min(closest(s,mid),closest(mid+1,e));// 递归求解
    int i,j,cnt=0;
    for(i=s;i<=e;i++){// 把 x 坐标在 px[mid].x-ans~px[mid].x+ans 范围内的点取出来

    if(px[i]->x>=px[mid]->x-ans&&px[i]->x<=px[mid]->x+ans)
            py[cnt++]=px[i];
    }
    sort(py,py+cnt,cmpy);//按 y 坐标排序
    for(i=0;i<cnt;i++){
        for(j=i+1;j<cnt;j++){//py 数组中的点是按照 y 坐标升序的
```

```
                if(py[j]->y-py[i]->y>=ans)
                    break;

        ans=min(ans,get_dis(py[i],py[j]));
            }
        }
        return ans;
}
int main(){
        int i,n;
        while(scanf("%d",&n)!=EOF){
            if(n==0)
                break;
            for(i=0;i<n;i++){

        scanf("%lf%lf%lf",&point[i].x,&point[i].y,
&point[i].z);
                px[i]=&point[i];
            }
            sort(px,px+n,cmpx);
            double distance=closest(0,n-1);
            printf("%.2lf\n",distance);
        }
        return 0;
}
```

先按 X 坐标将点分成两部分，分别算出两部分的最小距离（递归），然后选出分界线左右的点，按 y 坐标从小到大排序，遍历里面对每两个点，更新距离。

三角形

```
#include <math.h>
struct point{double x,y;};
struct line{point a,b;};

double distance(point p1,point p2){
        return
sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-
p2.y));
}

point intersection(line u,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;
}
//最小覆盖圆圆心
struct point circumcenter(struct point a,struct
point b,struct point c){
        struct line u,v;
        struct point o1,o2,o3,ans;
        double r1,r2,r3;
        bool p=0;
        o1.x=(b.x+c.x)/2;
        o1.y=(b.y+c.y)/2;
        r1=distance(b,c)/2;
        if(distance(o1,a)<=r1){
            ans=o1;
            p=1;
        }
        o2.x=(a.x+c.x)/2;
        o2.y=(a.y+c.y)/2;
        r2=distance(a,c)/2;
        if(distance(o2,b)<=r2){
            ans=o2;
            p=1;
        }
        o3.x=(b.x+a.x)/2;
        o3.y=(b.y+a.y)/2;
        r3=distance(b,a)/2;
        if(distance(o3,c)<=r3){
            ans=o3;
            p=1;
        }
        if(p==0){
            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);
        }
        else return ans;
}
//外心
point circumcenter(point a,point b,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(point a,point b,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(point a,point b,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(point a,point b,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(point a,point b,point c){
        point u,v;
        double
step=fabs(a.x)+fabs(a.y)+fabs(b.x)+fabs(b.y)+f
abs(c.x)+fabs(c.y);
        int i,j,k;
        u.x=(a.x+b.x+c.x)/3;
        u.y=(a.y+b.y+c.y)/3;
        while (step>1e-10)
                for (k=0;k<10;step/=2,k++)
                        for (i=-1;i<=1;i++)
                                for (j=-1;j<=1;j++){
                                        v.x=u.x+step*i;
                                        v.y=u.y+step*j;
                                        if
(distance(u,a)+distance(u,b)+distance(u,c)>dis
tance(v,a)+distance(v,b)+distance(v,c))
                                                u=v;
                                }
        return u;
}
```

凸包问题

```
#include<stdio.h>
#include<string.h>
#include<algorithm>
#include<math.h>
using namespace std;
struct point{
	double x;
	double y;
	double jiaodu;
}map[100005],zhan[100005];
bool que(struct point x,struct point y){
	if(x.jiaodu==y.jiaodu){
		if(x.y!=y.y) return x.y<y.y;
		else return x.x<y.x;
	}
	else return x.jiaodu<y.jiaodu;
};
int start;
int main(){
	int i,j,m,n,xuhao;
	double r,ymin,xmin,a,b,c,d,x,y,s;
	while(scanf("%d%lf",&n,&r)!=EOF){
		ymin=INT_MAX;
		xmin=INT_MAX;
		s=0;
		for(i=1;i<=n;i++){

	scanf("%lf%lf",&map[i].x,&map[i].y);
			if(map[i].y<ymin
||(map[i].y==ymin && xmin>map[i].x) ){
				ymin=map[i].y;
				xmin=map[i].x;
				xuhao=i;
			}
		}
		if(n>2){
			map[n+1]=map[xuhao];
			map[xuhao]=map[1];
			map[1]=map[n+1];
			for(i=2;i<=n;i++){
	map[i].jiaodu=atan2(map[i].y-map[1].y,m
ap[i].x-map[1].x);
			}
			sort(map+2,map+n+1,que);
			zhan[1]=map[1];
			zhan[2]=map[2];
			start=2;
			for(i=3;i<=n;i++){
				do{
					if(start>1){

	a=zhan[start].x-zhan[start-1].x;

	b=zhan[start].y-zhan[start-1].y;

	c=map[i].x-zhan[start].x;

	d=map[i].y-zhan[start].y;
						if(a*d-b*c<0)
							start--;
						else

	zhan[++start]=map[i];
					}
					else

	zhan[++start]=map[i];
				}while(a*d-b*c<0);
			}
			x=map[1].x;
			y=map[1].y;
			zhan[start+1]=zhan[1];
			for(i=1;i<=start;i++){

	s+=sqrt(  (zhan[i+1].x-zhan[i].x)*(zhan[i+1
].x-zhan[i].x)+(zhan[i+1].y-zhan[i].y)*(zhan[i+1
].y-zhan[i].y) );
			}
		}
		else if(n==2){

	s+=2*sqrt((map[1].x-map[2].x)*(map[1].
x-map[2].x)+(map[1].y-map[2].y)*(map[1].y-m
ap[2].y));
```

```
            }
            s+=3.1415926*2*r;
            printf("%.2lf\n",s);
        }
        return 0;
}
```

最小覆盖圆

```
#include<stdio.h>
#include<string.h>
#include<algorithm>
#include<math.h>
using namespace std;
struct point{
        double x;
        double y;
}map[100005],o,o1,o2,o3;
struct line{struct point a,b;};
struct point intersection(struct line u,struct line v){
        struct 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;
}
double dis(struct point x,struct point y){
        return
sqrt( (x.x-y.x)*(x.x-y.x)+(x.y-y.y)*(x.y-y.y) );
}
struct point circumcenter(struct point a,struct point b,struct point c){
        struct line u,v;
        struct point o1,o2,o3,ans;
        double r1,r2,r3;
        bool p=0;
        o1.x=(b.x+c.x)/2;
        o1.y=(b.y+c.y)/2;
        r1=dis(b,c)/2;
        if(dis(o1,a)<=r1){
                ans=o1;
                p=1;
        }
        o2.x=(a.x+c.x)/2;
        o2.y=(a.y+c.y)/2;
        r2=dis(a,c)/2;
        if(dis(o2,b)<=r2){
                ans=o2;
                p=1;
        }
        o3.x=(b.x+a.x)/2;
        o3.y=(b.y+a.y)/2;
        r3=dis(b,a)/2;
        if(dis(o3,c)<=r3){
                ans=o3;
                p=1;
        }
        if(p==0){
                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);
        }
        else return ans;
}
double maxzhi(double x,double y){
        if(x>y) return x;
        else return y;
}
int main(){
        int i,j,m,n,p1,p2,p3,xuhao,xuanze;
        double maxdis,r,r1,r2,r3;
        while(scanf("%d",&n)!=EOF && n){
                for(i=1;i<=n;i++)

        scanf("%lf%lf",&map[i].x,&map[i].y);
                if(n>2){
```

```
            p1=1;
            p2=2;
            p3=3;

    o=circumcenter(map[p1],map[p2],map[p
3]);

    r=maxzhi(dis(o,map[p1]),dis(o,map[p2]));
            while(1){
                maxdis=r;
                for(i=1;i<=n;i++){

    if(dis(o,map[i])>maxdis){

    maxdis=dis(o,map[i]);
                        xuhao=i;
                    }
                }
                xuanze=0;
                if(maxdis>r){

    o1=circumcenter(map[p2],map[p3],map[
xuhao]);

    r1=maxzhi(  dis(o1,map[p2]),dis(o1,map[
p3]) );

    if(dis(map[p1],o1)<=r1){
                        o=o1;
                        r=r1;
                        xuanze=1;
                    }

    o2=circumcenter(map[p1],map[p3],map[
xuhao]);

    r2=maxzhi(  dis(o2,map[p1]),dis(o2,map[
p3]) );

    if(dis(map[p2],o2)<=r2          &&
(r2<r||xuanze==0) ){
                        o=o2;
                        r=r2;
                        xuanze=2;
                    }

    o3=circumcenter(map[p1],map[p2],map[
xuhao]);

    r3=maxzhi(  dis(o3,map[p1]),dis(o3,map[
p2]) );

    if(dis(map[p3],o3)<=r3          &&
(r3<r||xuanze==0) ){
                        o=o3;
                        r=r3;
                        xuanze=3;
                    }
                    if(xuanze==1){
                        p1=xuhao;
                    }
                    else if(xuanze==2){
                        p2=xuhao;
                    }
                    else{
                        p3=xuhao;
                    }
                }
                else break;
            }
        }
        else{
            r=dis(map[1],map[2])/2;
        }
        printf("%.2lf\n",r);
    }
}
```

任意选 3 个点做覆盖圆，然后选出剩余点中最远的重新做可以覆盖这 4 个点的最小覆盖圆。直到所有点均被覆盖为止。