某 zzj の 表面模板

CSFNB！

常用工具

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

const double eps = 1e-10;
const double pi = 3.1415926535897932384626433832795;
const double eln = 2.718281828459045235360287471352;

typedef long long ll;
typedef unsigned long long ull;
typedef vector<int> vi;
typedef pair<int,int> pii;
typedef pair<ll,ll> pll;
#define IN freopen("in.txt","r",stdin)
#define OUT freopen("out.txt","w",stdout)
#define pr(x) cout<<#x<<": "<<x<<endl
#define lowbit(x) (x&(-x))
#define mp make_pair
#define pb push_back
#define fi first
#define se second
//#################################################################################
###########//
//                                                                    愉 悦 的 分 割 线
//
//#################################################################################
###########//


int main(){
	ios::sync_with_stdio(false);

	return 0;
}
//快速读入
inline void read(int& x){
	char ch = getchar();
	int flag = 1;
	while(ch < '0' || ch > '9'){
		if(ch == '-') flag = -1;
		ch = getchar();
	}
	x = 0;
	for(;ch >= '0' && ch <= '9';ch = getchar()) x=x*10+ch-'0';
	x *= flag;
}
inline void write(int x){
	if(x < 0){
		putchar('-');
		x *= -1;
	}
	char ch = x%10+'0';
	if(x > 10) write(x/10);
	putchar(ch);
}

//无去重离散化，新的下标从 1 开始，最大下标为_cnt
int idx[maxn],_cnt;
inline int get_id(int x){
	return lower_bound(idx+1,idx+_cnt+1,x) - idx;
}
void discretize(int* f,int size){
	for(int i = 1;i <= size;i++) idx[i] = f[i];
	sort(idx+1,idx+size+1);
	_cnt = size;
}

//去重离散化，新的下标从 1 开始，最大下标为_cnt
int idx[maxn],tmp_id[maxn],_cnt;
inline int get_id(int x){
	return lower_bound(idx+1,idx+_cnt+1,x) - idx;
}
void discretize(int* f,int size){
	_cnt = 0;
	for(int i = 1;i <= size;i++) tmp_id[i] = f[i];
	sort(tmp_id+1,tmp_id+size+1);
	for(int i = 1;i <= size;i++){
		idx[++_cnt] = tmp_id[i];
		while(i + 1 <= size && tmp_id[i] == tmp_id[i+1]) i++;
	}
}
//稳妥开根号
ll my_sqrt(ll x){
	ll ret = sqrt(x);
	while((ret+1)*(ret+1) <= x) ++ret;
	while( ret*ret > y) --ret;
	return ret;
}
```

数据结构

```
struct hash_node{
	int val1,val2;
	hash_node* next;
};
const int MOD_hash = 4e6+7;
has_node* hash_table[MOD_hash];
//仅返回 key 值是否存在用于去重等操作,可修改为真正的 hash 表
bool check_and_add(int v1,int v2){
	//将 v1 范围控制在可以直接访问的范围
	v1 = v1 % MOD_hash;
	hash_node** p = &hash_table[v1];
	while(*p){
		if((*p)->val1 == v1 && (*p)->val2 == v2) return false;
		p = &((*p)->next);
	}
	*p = new node;
	(*p)->val1 = v1;
	(*p)->val2 = v2;
	(*p)->next = 0;
	return false;
}
/*
顺便提供一些[1e6, 1e7]的大质数
1000003 1999993
2000003 2999999
3000017 3999971
4000037 4999999
5000011 5999993
6000011 6999997
7000003 7999993
8000009 8999993
9000011 9999991
*/
const int maxn = 5e5+10;
```

```
vector<int> G[maxn];//双向边
//子树大小，重儿子，深度，父亲
int size[maxn],son[maxn],deep[maxn],fa[maxn];
int dfn[maxn],top[maxn];//DFS 序，所在重链的顶端
int dfs_clock;//记得初始化，dfn 计算需要

void dfs1(int cur, int father, int depth) {
	size[cur] = 1,son[cur] = 0,deep[cur] = depth,fa[cur] = father;
	for(auto nx: G[cur]) {
		if(nx != father) {
			dfs1(nx, cur, depth + 1);
			size[cur] += size[nx];
			if(size[nx] > size[son[cur]]) {
				son[cur] = nx;
			}
		}
	}
}
void dfs2(int cur, int tp){
	top[cur] = tp,dfn[cur] = ++dfs_clock;
	if(son[cur]) dfs2(son[cur], cur, tp);//优先遍历重儿子
	for(auto nx: G[cur]) {
		if(nx != fa[cur] && nx != son[cur]) {
			dfs2(nx, nx);
		}
	}
}
int lca(int x, int y) {
	int t1 = top[x], t2 = top[y];
	while(t1 != t2) {
		//不在一个重链上，将深度大的往上跳
		if(deep[t1] < deep[t2]) {
			swap(x, y); swap(t1, t2);
		}
		x = fa[t1], t1 = top[x];
	}
	return deep[x] < deep[y] ? x : y;
}
//修改，对 u-v 这条链执行函数 f。查询也类似
void modify(int x, int y,void f(int,int)) {
	int t1 = top[x], t2 = top[y];
	while(t1 != t2) {
		if(deep[t1] < deep[t2]) {
			swap(x, y); swap(t1, t2);
```

```
            }
            f(dfn[t1], dfn[x]);
            x = fa[t1], t1 = top[x];
        }
        if(deep[x] > deep[y]) swap(x, y);
        f(dfn[x], dfn[y]);
}
//扫描线，例子为区间反转求 1 的个数
struct Line{
        ll x,y1,y2;
        Line(){}
        Line(ll x,ll y1,ll y2):x(x),y1(y1),y2(y2){}
}que[maxn];
bool cmp(Line a,Line b){
        return a.x < b.x;
}
int main(){
        ios::sync_with_stdio(false);
        return -1;
        int T,n,k;
        cin>>T;
        while(T){
            cin>>n>>k;
            memset(seg,0,sizeof(seg));
            memset(lazy,0,sizeof(lazy));
            int tot = 0;
            for(int i = 0;i < k;i++){
                int x1,y1,x2,y2;
                cin>>x1>>x2>>y1>>y2;
                que[tot++] = Line(x1-1,y1,y2);
                que[tot++] = Line(x2,y1,y2);
            }
            sort(que,que+tot,cmp);
            ll last = 0,ans = 0;
            for(int i = 0;i < tot;i++){
                ll tmp = query(1,1,n,1,n);
            //  cout<<tmp<<endl;
                if(que[i].x != 0) ans += (que[i].x - last) * query(1,1,n,1,n);
                add(1,1,n,que[i].y1,que[i].y2,1);
                last = que[i].x;
            }
            cout<<ans<<endl;
        }
}
```

```
typedef long long ll;
const int maxn = 1e6+10;
ll seg[maxn*4],lazy[maxn*4],a[maxn];

inline ll merge(ll a,ll b){
        return a + b;
        //return max(a,b);
        //return min(a,b);
}
inline ll addLazy(int o,int l,int r,ll x){
        seg[o] += x * (r-l+1);
        //seg[o] += x;
        /*区间反转
        if(x == 1) seg[o] = (r-l+1) - seg[o];
        lazy[o] ^= x;
        */
        lazy[o] += x;
}
inline void push_down(int o,int l,int r){
        int mid = (l + r) >> 1;
        addLazy(o<<1,l,mid,lazy[o]);
        addLazy(o<<1|1,mid+1,r,lazy[o]);
        lazy[o] = 0;
}
ll build(int o,int l,int r){
        if(l == r) return seg[o] = a[l];
        int mid = (l + r) >> 1;
        build(o<<1,l,mid);build(o<<1|1,mid+1,r);
        seg[o] = merge(seg[o<<1],seg[o<<1|1]);
}
ll add(int o,int l,int r,int L,int R,ll v){
        if(l >= L && r <= R) return addLazy(o,l,r,v);
        if(lazy[o] != 0) push_down(o,l,r);
        int mid = (l + r) >> 1;
        if(L <= mid) add(o<<1,l,mid,L,R,v);
        if(R > mid) add(o<<1|1,mid+1,r,L,R,v);
        seg[o] = merge(seg[o<<1],seg[o<<1|1]);
}
ll query(int o,int l,int r,int L,int R){
        if(L > R) return 0;
        if(l >= L && r <= R) return seg[o];
        if(lazy[o] != 0) push_down(o,l,r);
        int mid = (l + r) >> 1;
        ll ans = 0;//or inf
```

```
	if(L <= mid) ans = merge(ans,query(o<<1,l,mid,L,R));
	if(R > mid) ans = merge(ans,query(o<<1|1,mid+1,r,L,R));
	return ans;
}
//可修改为每个节点为 bitset 的线段树，修改数据类型 + add 函数 + combine 函数即可
const int maxn = 1e5+10;
ll a[maxn],seg[maxn*4];
//合并函数，取最大值、求和等等
inline ll combine(ll x,ll y){
	return max(x,y);
	//return min(x,y);
	//return x+y;
}
ll build(int o,int l,int r){
	if(l == r) return seg[o] = a[l];
	int mid = (l + r) / 2;
	build(o*2,l,mid);build(o*2+1,mid+1,r);
	seg[o] = combine(seg[o*2],seg[o*2+1]);
}
ll add(int o,int l,int r,int x,ll v){
	//更新的这一行需要根据需求修改
	if(l == r) return seg[o] = combine(seg[o],v);
	int mid = (l + r) / 2;
	if(x <= mid) add(o*2,l,mid,x,v);
	else add(o*2+1,mid+1,r,x,v);
	seg[o] = combine(seg[o*2],seg[o*2+1]);
}
ll query(int o,int l,int r,int L,int R){
	if(L > R) return 0;
	if(l >= L && r <= R) return seg[o];
	int mid = (l + r) / 2;
	ll ans = 0;
	if(L <= mid) ans = combine(ans,query(o*2,l,mid,L,R));
	if(R > mid) ans = combine(ans,query(o*2+1,mid+1,r,L,R));
	return ans;
}
```

图论

```
void dijkstra(int st,int n){
	priority_queue<pll> q;
	for(int i = 1;i <= n;i++) dist[i] = 1e18+10;
	dist[st] = 0;q.push(mp(0,st));
	while(!q.empty()){
		pll cur = q.top();q.pop();
		if(dist[cur.se] != -cur.fi) continue;
		for(pll p : G[cur.se]){
			if(dist[p.fi] > dist[cur.se] + p.se){
				dist[p.fi] = dist[cur.se] + p.se;
				q.push(mp(-dist[p.fi],p.fi));
			}
		}
	}
}
bool spfa(int st,int n){
	ll cnt[maxn] = {0};
	queue<pll> q;
	for(int i = 1;i <= n;i++) dist[i] = 1e18+10;
	dist[st] = 0;q.push(mp(0,st));
	while(!q.empty()){
		pll cur = q.front();q.pop();
		if(++cnt[cur.se] > n + 1) return false;
		for(pll p : G[cur.se]){
			if(dist[p.fi] > dist[cur.se] + p.se){
				dist[p.fi] = dist[cur.se] + p.se;
				q.push(mp(dist[p.fi],p.fi));
			}
		}
	}
	return true;
}
//费用流
const int inf = 0x3f3f3f3f;
const int mm = 111111;
const int maxn = 999;
int node,src,dest,edge;
int ver[mm],flow[mm],cst[mm],nxt[mm];
int head[maxn],work[maxn],dis[maxn],q[maxn];
int tot_cost;
void prepare(int _node,int _src,int _dest){
	node=_node,src=_src,dest=_dest;
	for(int i=0; i<node; ++i)head[i]=-1;
	edge=0;
	tot_cost = 0;
}
void add_edge(int u,int v,int c,int cost){
        ver[edge]=v,flow[edge]=c,nxt[edge]=head[u],cst[edge]=cost,head[u]=edge++;
        ver[edge]=u,flow[edge]=0,nxt[edge]=head[v],cst[edge]=-cost,head[v]=edge++;
}
int ins[maxn];
int pre[maxn];
bool Dinic_spfa(){
        memset(ins,0,sizeof(ins));
        memset(dis,inf,sizeof(dis));
        memset(pre,-1,sizeof(pre));
        queue<int> Q;
        //int i,u,v,l,r=0;
        Q.push(src);
        dis[src] = 0,ins[src] = 1;
        pre[src] = -1;
        while(!Q.empty()){
                int u = Q.front();Q.pop();
                ins[u] = 0;
                for(int e = head[u];e != -1;e = nxt[e]){
                        int v = ver[e];
                        if(!flow[e]) continue;
                        if(dis[v] > dis[u] + cst[e]){
                                dis[v] = dis[u] + cst[e];
                                pre[v] = e;
                                if(!ins[v]) ins[v] = 1,Q.push(v);
                        }
                }
        }
        return dis[dest] < inf;
}
int Dinic_flow(){
        int i,ret=0,delta=inf;
        while(Dinic_spfa()){
                for(int i=pre[dest];i != -1;i = pre[ver[i^1]])
                        delta = min(delta,flow[i]);
                for(int i=pre[dest];i != -1;i = pre[ver[i^1]])
                        flow[i] -= delta,flow[i^1] += delta;
                ret += delta;
                tot_cost += dis[dest]*delta;
        }
        return ret;
}
```

算法：（矩阵快速幂，模拟退火，LIS）

```
/*
只返回结果值，需要区间求最大值线段树 + 离散化
可修改转移方程，实现求最大上升序列和
*/
ll LIS(int* f,int len){
	ll ret = 0;
	//离散化，可以在仅主函数内进行一次
	discretize(f,len);
	memset(seg,0,sizeof(seg));
	for(int i = 1;i <= len;i++){
		//不是严格小于需要修改询问，方程可能需要修改
		ll tmp = query(1,1,_cnt,1,get_id(f[i])-1) + 1;
		add(1,1,_cnt,get_id(f[i]),tmp);
		ret = max(ret,tmp);
	}
	return ret;
}
/*
在以上的基础上输出方案
由于线段树上的节点是一个 pll，需修改线段树保证输出字典序最小方案
输出字典序最小方案，pair 里面的 second 存的是序号取负，这样可以直接对 pair 取 max，
若是取 min 等，需修改
*/
int last[maxn];
vi LIS(int* f,int len){
	ll ret = 0,mark;
	discretize(f,len);
	memset(seg,0,sizeof(seg));
	for(int i = 1;i <= len;i++){
		//不是严格小于需要修改询问，方程可能需要修改
		pll tmp = query(1,1,_cnt,1,get_id(f[i])-1);
		ll val = tmp.fi + 1;
		last[i] = -tmp.se;
		add(1,1,_cnt,get_id(f[i]),mp(val,-i));
		if(val > ret){
			ret = val;
			mark = i;
		}
	}
	stack<int> s;
	while(mark != 0){
		s.push(mark);
		mark = last[mark];
	}
	vi retv;
	while(!s.empty()){
		retv.pb(s.top());
		s.pop();
	}
	return retv;
}
//TODO:以下模板还未测试过
/*
对数据进行处理之后，选取权值最大字典序最小的 x1 < x2，y1 < y2 的序列方案输出
若是 x1 <= x2，可排序后归约为正常的 LIS
x 和 y 需要是全局变量，大小为 len，下标从 1 开始
*/
int ord[maxn],last[maxn];
bool cmp(int a,int b){
	return x[a] < x[b];
}
//线段树中的 first 是 val，second 是编号取负
vi LIS(int len){
	//x 取值可能是-1 时需要修改
	ll ret = 0,mark,lastx = -1;
	for(int i = 1;i <= len;i++) ord[i] = i;
	sort(ord+1,ord+len+1,cmp);
	discretize(y,len);
	memset(seg,0,sizeof(seg));
	queue<pair<ll,pll> > q;
	for(int i = 1;i <= len;i++){
		if(lastx != x[ord[i]]){
			while(!q.empty()){
				add(1,1,_cnt,q.front().fi,q.front().se);
				q.pop();
			}
			lastx = x[ord[i]];
		}
		pll tmp = query(1,1,_cnt,1,get_id(y[ord[i]])-1);
		ll val = tmp.fi + 1;
		last[i] = -tmp.se;
		q.push(mp(get_id(y[ord[i]]), mp(val,-ord[i])));
		if(val > ret){
			ret = val;
			mark = ord[i];
		}
	}
	stack<int> s;
	while(mark != 0){
		s.push(mark);
		mark = last[mark];
	}
	vi retv;
	while(!s.empty()){
		retv.pb(s.top());
		s.pop();
	}
	return retv;
}
/*
上面那个占空间太大了,太复杂，写一个不输出方案只输出答案的版本
*/
int ord[maxn];
bool cmp(int a,int b){
	return x[a] < x[b];
}
ll LIS(int len){
	//x 取值可能是-1 时需要修改
	ll ret = 0,lastx = -1;
	for(int i = 1;i <= len;i++) ord[i] = i;
	sort(ord+1,ord+len+1,cmp);
	discretize(y,len);
	memset(seg,0,sizeof(seg));
	queue<pll> q;
	for(int i = 1;i <= len;i++){
		if(lastx != x[ord[i]]){
			while(!q.empty()){
				add(1,1,_cnt,q.front().fi,q.front().se);
				q.pop();
			}
			lastx = x[ord[i]];
		}
		ll tmp = query(1,1,_cnt,1,get_id(y[ord[i]])-1) + 1;
		q.push(mp(get_id(y[ord[i]]),tmp));
		ret = max(ret,tmp);
	}
	return ret;
}
```

```
//矩阵快速幂
#include <bits/stdc++.h>
using namespace std;

const int MOD = 1e9+7;
typedef long long ll;

struct Mat{
	int size;
	vector<vector<ll> > a;
	Mat(int size = 3):size(size){}
	void setZero(){
		for(int i = 0;i < size;i++){
			vector<ll> tmp(size);
			for(int j = 0;j < size;j++){
				tmp[j] = 0;
			}
			a.push_back(tmp);
		}
	}
	void setUnit(){
		setZero();
		for(int i = 0;i < size;i++) a[i][i] = 1;
	}
	vector<ll>& operator[](size_t n){
		return a[n];
	}
	Mat operator*(Mat& tar){
		Mat ret(size);
		ret.setZero();
		for(int i = 0;i < size;i++){
			for(int j = 0;j < size;j++){
				for(int k = 0;k < size;k++){
					ret[i][j] = (ret[i][j] + a[i][k] * tar[k][j])%MOD;
				}
			}
		}
		return ret;
	}
};

Mat matPow(Mat a,ll b){
	Mat ret(a.size);
	ret.setUnit();
	while(b){
		if(b & 1) ret = ret * a;
		a = a * a;
		b >>= 1;
	}
	return ret;
}

Mat init(){
	Mat ret(9);
	ret.setZero();
	ret[0][1] = ret[0][2] = 1;
	ret[1][4] = ret[1][5] = 1;
	ret[2][7] = ret[2][8] = 1;
	ret[3][0] = ret[3][1] = 1;
	ret[4][3] = ret[4][5] = 1;
	ret[5][6] = ret[5][7] = ret[5][8] = 1;
	ret[6][0] = ret[6][2] = 1;
	ret[7][3] = ret[7][4] = ret[7][5] = 1;
	ret[8][6] = ret[8][7] = 1;
	return ret;
}

int main(){
	ios::sync_with_stdio(false);
	ll n,T;
	cin>>T;
	while(T--){
		cin>>n;
		if(n == 1){
			cout<<3<<endl;
			continue;
		}
		if(n == 2){
			cout<<9<<endl;
			continue;
		}
		Mat M = init();
		M = matPow(M,n-2);
		ll ans = 0;
		for(int i = 0;i < 9;i++){
			for(int j = 0;j < 9;j++)
				ans = (ans + M[i][j]) % MOD;
```

```
        }
        cout<<ans<<endl;
    }
    return 0;
}
/*
模拟退火例题 1：
有 n 个重物，每个重物系在一条足够长的绳子上。每条绳子自上而下穿过桌面上的洞，然后系在一起。图中 X 处就是公共的绳结。
假设绳子是完全弹性的（不会造成能量损失），桌子足够高（因而重物不会垂到地上），且忽略所有的摩擦。
问绳结 X 最终平衡于何处。

可直接随机选点，每次往力的方向移动
也可根据评估函数，让每个物体的势能和最小

注意不同的题目，参数 T 的设置需要根据情况判断
*/
const double delta = 0.99;
Vector getNext(Point *p,Point now,int n){
    Vector ret = Point(0,0);
    for(int i = 0;i < n;i++){
        ret = ret + unit(p[i]-now)*w[i];
    }
    return ret;
}
Point work(Point* p,int n){
    double t = 1,min_dis = 1e18;
    Point now = p[0],ans = p[0];
    while(t > eps){
        Vector dv = getNext(p,now,n);
        double l = Length(dv);
        if(min_dis > l){
            min_dis = l;
            ans = now;
        }
        now.x = now.x + dv.x * t;
        now.y = now.y + dv.y * t;
        t *= delta;
    }
    return ans;
}
/*
模拟退火例题 2：
```

```
最小球包含
精度不能达到很高
*/
const double delta = 0.98;
//寻找离当前圆心最远的点
int getPoint(Point *p,Point now,int n){
	int res = -1;
	double max_dis = 0,pre= 0;
	for(int i = 0;i < n;i++){
		max_dis = max(max_dis,Length(p[i]-now));
		if(max_dis != pre) res = i;
		pre = max_d;
	}
	return res;
}
//此处 t 的初始值设为 1 能过，设为 0.5 就 WA，目前还不是很懂
Point work(Point* p,int n){
	double t = 1,min_r = 1e18;
	Point now = p[0],ans = p[0];
	while(t > eps){
		int i = getPoint(p,now,n);
		now.x = now.x + (p[i].x - now.x) * t;
		now.y = now.y + (p[i].y - now.y) * t;
		now.z = now.z + (p[i].z - now.z) * t;
		if(min_dis > r){
			min_dis = r;
			ans = now;
		}
		t *= delta;
	}
	return ans;
}
```

数位 DP

```
//计算[1-x]有多少个数最多只有 3 个位置不是 0
int num[20],len;
ll cal(int x,bool limit,int last){
	if(last == 0 || x>= len) return 1;
	if(limit&&num[x] == 0) return cal(x+1,limit,last);
	ll ans = 0;
	if(limit){
		//1 - num[x]-1
		if(num[x] != 1) ans += (num[x]-1) * cal(x+1,false,last - 1);
		//0
		if(num[x] != 0) ans += cal(x+1,false,last);
		ans += cal(x+1,true,last-1);
	} else{
		ans += 9 * cal(x+1,false,last-1);
		ans += cal(x+1,false,last);
	}
	return ans;
}
ll solve(ll x){
	ll tmp[20],ps = 0;
	while(x){
		tmp[ps++] = x%10;
		x /= 10;
	}
	for(int i = 0;i < ps;i++){
		num[i] = tmp[ps-1-i];
	}
	len = ps;
	return cal(0,true,3);
}
```

求组合数

```
long long fac[400],inv[400];
const int MOD = 1e9+7;
long long pow_mod(long long x, long long n, long long mod){
	long long res=1;
	while(n>0){
		if(n&1)res=res*x%mod;
		x=x*x%mod;
		n>>=1;
	}
	return res;
}
void init(){
	fac[0] = 1;
	for(int i = 1;i <= 300;++i){
		fac[i] = fac[i-1]*i%MOD;
	}
	for(int i = 0;i <= 300;++i){
		inv[i] = pow_mod(fac[i],MOD-2,MOD);
	}
}
long long C(long long n,long long m){
	long long ans = fac[n];
	ans = ans * inv[m] % MOD;
	ans = ans * inv[n-m] % MOD;
	return ans;
}
```

Manacher

```
//重要性质：一个字符串最多只有 n 个本质不同的回文子串
int p[maxn*2];
int manacher(string s_source){
	//预处理左加#右加$，##a#b#c#$
	string s = "##";
	for(int i = 0;i < s_source.size();i++){
		s += s_source[i];
		s += '#';
	}
	s += '$';
	int max_len = -1,mx = 0,id;
	for(int i = 1;i < s.size() - 1;i++){
		p[i] = i < mx ? min(p[2 * id - i], mx - i) : 1;
		while(s[i - p[i]] == s[i + p[i]]) p[i]++;//在这一行修改可统计所有的回文子串
		if (mx < i + p[i]) mx = i + p[i],id = i;
		max_len = max(max_len, p[i] - 1);
	}
	return max_len;
}
```

计算几何

```
struct Point{
	ll x,y,id;
	Point(ll _x = 0,ll _y = 0):x(_x),y(_y){}
};
```

```
bool operator<(const Point& a,const Point& b){
        return (a.x < b.x) || (a.x == b.x && a.y < b.y);
}
Point operator-(Point a,Point b){
        return Point(a.x-b.x, a.y-b.y);
}
Point operator+(Point a,Point b){
        return Point(a.x+b.x, a.y+b.y);
}
ll operator*(Point a,Point b){
        return a.x*b.x + a.y*b.y;
}
Point operator*(Point a,ll b){
        return Point(a.x*b, a.y*b);
}
inline ll Cross(Point a,Point b){
        return a.x*b.y - a.y*b.x;
}
inline double Length(Point a){
        return sqrt((double)(a*a));
}
inline double Angle(Point a,Point b){
        return acos((double)(a * b) / Length(a) / Length(b));
}
//不损失精度判断线段规范相交(不含端点)
/若要判断线段是否有点在多边形内部，最好缩多边形，判任一公共点，
//或者把线段端点往里缩一下，同时取中点，check 一下这三个点是不是在多边形内部
bool isSegmentsIntersection(Point A,Point B,Point C,Point D){
        //跨立试验
        if(Cross(C-A,D-A) * Cross(C-B,D-B) >= 0) return false;
        if(Cross(A-C,B-C) * Cross(A-D,B-D) >= 0) return false;
        //快速排斥试验
        if(min(max(A.x,B.x),max(C.x,D.x)) < max(min(A.x,B.x),min(C.x,D.x))) return false;
        if(min(max(A.y,B.y),max(C.y,D.y)) < max(min(A.y,B.y),min(C.y,D.y))) return false;
        return true;
}
//点在线段上(//不含端点)
bool isPointOnSegment(Point P,Point a,Point b){
        if(P == a || P == b) return true;
        //if(p == a || p == b) return false;
        return Cross(a-P,b-P) == 0 && (a-P)*(b-P) < 0;
}
//判断两条线段是否有公共点
bool isSegmengtsCrash(Point A,Point B,Point C,Point D){
    if( isPointOnSegment(A,C,D) || isPointOnSegment(B,C,D) ||
        isPointOnSegment(C,A,B) || isPointOnSegment(D,A,B)) return true;
    if(Cross(B-A,D-C) == 0) return false;//共线
    return isSegmentsIntersection(A,B,C,D);//判断线段规范相交
}
struct Point{
    double x,y;ll id;
    Point(double _x = 0,double _y = 0):x(_x),y(_y){}
};
bool operator<(const Point& a,const Point& b){
    return (a.x < b.x) || (a.x == b.x && a.y < b.y);
}
int dcmp(double x){
    if(fabs(x) < eps) return 0;else return x < 0 ? -1 : 1;
}
bool operator==(const Point& a,const Point& b){
    return dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y) == 0;
}
Point operator-(Point a,Point b){
    return Point(a.x-b.x, a.y-b.y);
}
Point operator+(Point a,Point b){
    return Point(a.x+b.x, a.y+b.y);
}
double operator*(Point a,Point b){
    return a.x*b.x + a.y*b.y;
}
Point operator*(Point a,double b){
    return Point(a.x*b, a.y*b);
}
inline double Cross(Point a,Point b){
    return a.x*b.y - a.y*b.x;
}
inline double Length(Point a){
    return sqrt(a.x*a.x + a.y*a.y);
}
inline double Length2(Point a){
    return a.x*a.x + a.y*a.y;
}
//单位化向量 ，若是零向量直接返回
Point unit(Point a){
    double l = Length(a);
    if(l < eps) return a;
    return Point(a.x/l,a.y/l);
```

```
}
inline double Angle(Point a,Point b){
        return acos(a * b / Length(a) / Length(b));
}
//有向面积
double Area2(Point a,Point b,Point c){
        return Cross(b-a,c-a);
}
Point rotate(Point a,double rad){
        return Point(a.x*cos(rad)-a.y*sin(rad), a.x*sin(rad)+a.y*cos(rad));
}
//求向量 A 的左转法向量
Point normal(Point a){
        return Point(-a.y,a.x);
}
//求单位左转法向量，调用前请保证 A 不是零向量
Point unitNormal(Point a){
        double l = Length(a);
        return Point(-a.y/l,a.x/l);
}
//不损失精度判断线段规范相交(不含端点)
//若要判断线段是否有点在多边形内部，最好缩多边形，判任一公共点，
//或者把线段端点往里缩一下，同时取中点，check 一下这三个点是不是在多边形内部
bool isSegmentsIntersection(Point A,Point B,Point C,Point D){
        //跨立试验
        if(Cross(C-A,D-A) * Cross(C-B,D-B) >= 0) return false;
        if(Cross(A-C,B-C) * Cross(A-D,B-D) >= 0) return false;
        //快速排斥试验
        if(min(max(A.x,B.x),max(C.x,D.x)) < max(min(A.x,B.x),min(C.x,D.x))) return false;
        if(min(max(A.y,B.y),max(C.y,D.y)) < max(min(A.y,B.y),min(C.y,D.y))) return false;
        return true;
}
//点在线段上(//不含端点)
bool isPointOnSegment(Point P,Point a,Point b){
        if(P == a || P == b) return true;
        //if(p == a || p == b) return false;
        return dcmp(Cross(a-P,b-P)) == 0 && dcmp((a-P)*(b-P)) < 0;
}
//判断两条线段是否有公共点
bool isSegmengtsCrash(Point A,Point B,Point C,Point D){
        if( isPointOnSegment(A,C,D) || isPointOnSegment(B,C,D) ||
                isPointOnSegment(C,A,B) || isPointOnSegment(D,A,B)) return true;
        if(dcmp(Cross(B-A,D-C)) == 0) return false;//共线
        return isSegmentsIntersection(A,B,C,D);//判断线段规范相交
```

```
}
Point midPoint(Point a,Point b){
        return Point((a.x+b.x)*0.5,(a.y+b.y)*0.5);
}
//------------------线段相关内容--------------------
//有向直线
struct Line{
        Point p1,p2;//直线上两点，从 p1 到 p2，左边是半平面
        double ang;//极角，从 x 正半轴转到 v 所需的角(弧度)
        Line(){}
        Line(Point p1, Point p2):p1(p1),p2(p2){
                ang = atan2(p2.y-p1.y, p2.x-p1.x);
        }
        bool operator < (const Line& L) const{ //半平面交需要的排序函数
                return ang < L.ang;
        }
};
//直线相交，使用前保证有唯一交点，cross(v,w)非 0
Point getLineIntersection(Point A, Point B, Point C, Point D){
        Point u = A - C, v = B - A, w = D - C;
        double t = Cross(w, u) / Cross(v, w);
        return A + v * t;
}
Point getLineIntersection(Line L1, Line L2){
        Point u = L1.p1 - L2.p1, v = L1.p2 - L1.p1, w = L2.p2 - L2.p1;
        double t = Cross(w, u) / Cross(v, w);
        return L1.p1 + v * t;
}
//点到直线距离
double distanceToLine(Point P,Line L){
        Point v1 = L.p2 - L.p1, v2 = P - L.p1;
        return fabs(Cross(v1,v2)) / Length(v1);//不取绝对值就是有向距离
}
//----------------多边形相关内容----------------------
typedef vector<Point> polygon;
//-----------------圆相关内容-----------------------
struct Circle{
        Point o;
        double r;
        Circle(Point o,double r):o(o),r(r){}
        Point point(double rad){
                return Point(o.x+cos(rad)*r,o.y+sin(rad)*r);
        }
};
```

```
//给定两点作为直径获取圆
Circle getCircle(Point a,Point b){
	return Circle((a+b)*0.5,Length(a-b)*0.5);
}
//给予三个点，求外接圆
Circle Getcir(Point A,Point B,Point C){
	double a = 2*(B.x - A.x);
	double b = 2*(B.y - A.y);
	double c = (B.x*B.x+B.y*B.y) - (A.x*A.x+A.y*A.y);
	double d = 2*(C.x-B.x);
	double e = 2*(C.y-B.y);
	double f = (C.x*C.x + C.y*C.y) - (B.x*B.x + B.y*B.y);
	double x = (b*f-e*c)/(b*d-e*a);
	double y = (d*c-a*f)/(b*d-e*a);
	double r = sqrt((x-A.x)*(x-A.x) + (y-A.y)*(y-A.y));
	Point ans(x,y);
	return Circle(ans,r);
}
//包含三个点的面积最小的圆(注意，不是外接圆)
Circle getMinCircle(Point a,Point b,Point c){
	if(dcmp(Cross(b-a,c-a)) == 0){
		//三点共线
		if (dcmp(Length(a-b)+Length(b-c)-Length(a-c))==0) return getCircle(a,c);
		if (dcmp(Length(b-a)+Length(a-c)-Length(b-c))==0) return getCircle(b,c);
		if (dcmp(Length(a-c)+Length(c-b)-Length(a-b))==0) return getCircle(a,b);
	} else{
		if((b-a)*(c-a) <= 0) return getCircle(b,c);
		if((a-b)*(c-b) <= 0) return getCircle(a,c);
		if((a-c)*(b-c) <= 0) return getCircle(a,b);
		Point m1 = midPoint(a,b), m2 = midPoint(a,c);
		Line L1 = Line(m1,m1 + normal(b-a));
		Line L2 = Line(m2,m2 + normal(c-a));
		Point o = getLineIntersection(L1,L2);
		return Circle(o,Length(a-o));
	}
}
//点在圆内(不含边界是<)
bool pointInCircle(Point a,Circle c){
	return dcmp(Length2(a-c.o)-c.r*c.r) <= 0;
}
/*
Andrew 算法基于水平序求凸包
输入点的数组 p，点的个数 n，布尔数组为 1 表示跳过该编号的点；返回凸包点的个数，凸包的点存在 ch 数组里
```

```
直线上的点也要算的话，把两个<=改成<
精度要求高时建议使用三态函数
warning：下标从 0 开始
warning: 如果允许计算直线上的多个点，同时可以会是退化多边形的话，用下面的另一个版
本
*/
int convexHull(Point* p,bool* check,int n,Point* ch,Polygon& poly){
	sort(p,p+n);
	int m = 0;
	for(int i = 0;i < n;i++){
		if(check[p[i].id]) continue;
		while(m > 1 && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
		ch[m++] = p[i];
	}
	int k = m;
	for(int i = n-2;i >= 0;i--){
		if(check[p[i].id]) continue;
		while(m > k && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
		ch[m++] = p[i];
	}
	if(m > 1) m--;
	poly.clear();
	for(int i = 0;i < m;i++) poly.push_back(ch[i]);
	return m;
}
/*
版本 2，用于处理特殊情况。
特殊情况：需要计算直线上的点，同时可能会有退化成直线的多边形
*/
int convexHull(Point* p,bool* check,int n,Point* ch,Polygon& poly){
	sort(p,p+n);
	int m = 0,st = n;
	bool vis[n] = {0};
	for(int i = 0;i < n;i++){
		if(check[p[i].id]) continue;
		st = min(st,i);
		while(m > 1 && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) vis[ch[--m].id] = false;
		vis[p[i].id] = true;
		ch[m++] = p[i];
	}
	int k = m;
	for(int i = n-2;i >= 0;i--){
		if(check[p[i].id]) continue;
		if(i != st && vis[p[i].id]) continue;
		while(m > k && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
		ch[m++] = p[i];
	}
	if(m > 1) m--;
	poly.clear();
	for(int i = 0;i < m;i++) poly.push_back(ch[i]);
	return m;
}
/*
//点 p 在直线 L 的左边(不包括线上)  修改  >=
bool onLeft(Point p,Line L){
	return Cross(L.p2, p-L.p1) > 0;
}
//半平面交，不能计算退化的多边形。但是可以将每个半平面略微扩大求得交为单点或者线
的情况
//返回半平面交后的多边形的顶点数，多边形存在 poly 种
int halfplaneIntersection(Line* L,int n,Polygon& poly){
	sort(L,L+n);                    //按极角排序
	int first,last;                 //双端队列指针
	Point *p = new Point[n]; //p[i]为 q[i]和 q[i+1]的交点
	Line *q = new Line[n];          //双端队列
	q[first=last=0] = L[0];         //双端队列初始化
	for(int i = 1;i < n;i++){
		while(first < last && !onLeft(p[last-1], L[i])) last--;
		while(first < last && !onLeft(p[first], L[i])) first++;
		q[++last] = L[i];
		if(fabs(Cross(q[last].v, q[last-1].v)) < eps){
			//两向量平行且同向，取内侧的一个
			last--;
			if(onLeft(L[i].P, q[last])) q[last] = L[i];
		}
		if(first < last) p[last-1] = getLineIntersection(q[last-1],q[last]);
	}
	while(first < last && !onLeft(p[last-1], q[first])) last--;
	//此处要注意，若可能会出现无界区域，应在运行前手动加入四个特殊半平面将区域框
起来
	//删除无用平面(*)
	if(last - first <= 1) return 0;     //空集
	p[last] = getLineIntersection(q[last],q[first]); //计算首尾两个半平面的交点
	//从双端队列把答案复制到 poly 中
	poly.size = last - first + 1;
	poly.ps.clear();
	for(int i = first;i <= last;i++) poly.ps.pb(p[i]);
	return poly.size;
```

```
}
*/
//二维最小圆覆盖，随机增量法
Circle minCircle(Point *p, int n){
	random_shuffle(p,p+n);
	Circle cur = Circle(p[0],0);
	for(int i = 1;i < n;i++){
		if(pointInCircle(p[i],cur)) continue;
		cur = Circle(p[i],0);
		for(int j = 0;j < i;j++){
			if(pointInCircle(p[j],cur)) continue;
			cur = getCircle(p[i],p[j]);
			for(int k = 0;k < j;k++){
				if(pointInCircle(p[k],cur)) continue;
				cur = getMinCircle(p[i],p[j],p[k]);
			}
		}
	}
	return cur;
}
```

1.有多边形阻挡的情况下，求两点是否能直接连线，最好将多边形向内缩很小的距离，然后判断线段是否有任一公共点（不是判规范相交）