typedef double DB;

struct data{

LL pos[10];

int id;

} T[N] , op , point;

int split[N],now,n,demension;

bool use[N];

LL ans,id;

DB var[10];

bool cmp(data a,data b){

return a.pos[split[now]]<b.pos[split[now]];

}

void build(int L,int R){

if(L>R) return;

int mid=(L+R)>>1;

//求出 每一维 上面的方差

for(int pos=0; pos<demension; pos++){

DB ave=var[pos]=0.0;

for(int i=L; i<=R; i++)

ave+=T[i].pos[pos];

ave/=(R-L+1);

for(int i=L; i<=R; i++)

var[pos]+=(T[i].pos[pos]-ave)\*(T[i].pos[pos]-ave);

var[pos]/=(R-L+1);

}

//找到方差最大的那一维，用它来作为当前区间的 split\_method

split[now=mid]=0;

for(int i=1; i<demension; i++)

if(var[split[mid]]<var[i]) split[mid]=i;

//对区间排排序，找到中间点

nth\_element(T+L,T+mid,T+R+1,cmp);

build(L,mid-1);

build(mid+1,R);

}

void query(int L,int R)

{

if(L>R) return;

int mid=(L+R)>>1;

//求出目标点 op 到现在的根节点的距离

LL dis=0;

for(int i=0; i<demension; i++)

dis+=(op.pos[i]-T[mid].pos[i])\*(op.pos[i]-T[mid].pos[i]);

//如果当前区间的根节点能够用来更新最近距离，并且 dis 小于已经求得的 ans

if(!use[T[mid].id] && dis<ans){

ans=dis; //更新最近距离

point=T[mid]; //更新取得最近距离下的点

id=T[mid].id; //更新取得最近距离的点的 id

}

//计算 op 到分裂平面的距离

LL radius=(op.pos[split[mid]]-T[mid].pos[split[mid]])\*(op.pos[split[mid]]-T[mid].pos[split[mid]]);

//对子区间进行查询

if(op.pos[split[mid]]<T[mid].pos[split[mid]]){

query(L,mid-1);

if(radius<=ans) query(mid+1,R);

}else{

query(mid+1,R);

if(radius<=ans) query(L,mid-1);

}

}

int main(){

while(scanf("%d%d",&n,&demension)!=EOF){

//读入 n 个点

for(int i=1; i<=n; i++){

for(int j=0; j<demension; j++)

scanf("%I64d",&T[i].pos[j]);

T[i].id=i;

}

build(1,n); //建树

int m,q;

scanf("%d",&q); // q 个询问

while(q--){

memset(use,0,sizeof(use));

for(int i=0; i<demension; i++)

scanf("%I64d",&op.pos[i]);

scanf("%d",&m);

printf("the closest %d points are:\n",m);

while(m--){

ans=(((LL)INT\_INF)\*INT\_INF);

query(1,n);

for(int i=0; i<demension; i++){

printf("%I64d",point.pos[i]);

if(i==demension-1) printf("\n");

else printf(" ");

}

use[id]=1;

}

}

}

return 0;

}