//KD-Tree（原题来自bzoj2648）

//二维KDTree复杂度O(n^1.5)

int n,m,root;

namespace KDTree

{

struct node

{

int lson,rson;

int d[2],minn[2],maxn[2];

}a[1000048];

inline int myabs(int x) {return x>=0?x:-x;}

inline void update(int cur,int k)

{

a[cur].minn[0]=min(a[cur].minn[0],a[k].minn[0]);

a[cur].minn[1]=min(a[cur].minn[1],a[k].minn[1]);

a[cur].maxn[0]=max(a[cur].maxn[0],a[k].maxn[0]);

a[cur].maxn[1]=max(a[cur].maxn[1],a[k].maxn[1]);

}

int cmpd;

inline bool cmp(node x,node y)

{

if (x.d[cmpd]!=y.d[cmpd]) return x.d[cmpd]<y.d[cmpd];

return x.d[cmpd^1]<y.d[cmpd^1];

}

inline int build(int left,int right,int D)

{

int mid=(left+right)>>1;cmpd=D;

nth\_element(a+left,a+mid,a+right+1,cmp);

a[mid].minn[0]=a[mid].maxn[0]=a[mid].d[0];

a[mid].minn[1]=a[mid].maxn[1]=a[mid].d[1];

if (left!=mid) a[mid].lson=build(left,mid-1,D^1);

if (right!=mid) a[mid].rson=build(mid+1,right,D^1);

if (a[mid].lson) update(mid,a[mid].lson);

if (a[mid].rson) update(mid,a[mid].rson);

return mid;

}

inline void Insert(int pos)

{

int D=0,cur=root;

while (true)

{

update(cur,pos);

if (a[pos].d[D]<a[cur].d[D])

{

if(!a[cur].lson) {a[cur].lson=pos;return;} else {cur=a[cur].lson,D^=1;}

}

else

{

if(!a[cur].rson){a[cur].rson=pos;return;} else {cur=a[cur].rson,D^=1;}

}

}

}

//一种鬼畜的估价函数

inline int getdist(int cur,int x,int y)

{

int res=0;

if (x<a[cur].minn[0]) res+=a[cur].minn[0]-x;

if (x>a[cur].maxn[0]) res+=x-a[cur].maxn[0];

if (y<a[cur].minn[1]) res+=a[cur].minn[1]-y;

if (y>a[cur].maxn[1]) res+=y-a[cur].maxn[1];

return res;

}

inline void query(int cur,int x,int y)

{

ans=min(ans,myabs(x-a[cur].d[0])+myabs(y-a[cur].d[1]));

int dl=(a[cur].lson?getdist(a[cur].lson,x,y):INF);

int dr=(a[cur].rson?getdist(a[cur].rson,x,y):INF);

if (dl<dr)

{

if (dl<ans) query(a[cur].lson,x,y);

if (dr<ans) query(a[cur].rson,x,y);

}

else

{

if (dr<ans) query(a[cur].rson,x,y);

if (dl<ans) query(a[cur].lson,x,y);

}

}

}

int main ()

{

//这个模板是建树，插入，查找距某个点曼哈顿距离最近的点的距离

using namespace KDTree;

int i,type,x,y;

n=getint();m=getint();

for (i=1;i<=n;i++) a[i].d[0]=getint(),a[i].d[1]=getint();

root=KDTree::build(1,n,0);

for (i=1;i<=m;i++)

{

type=getint();

if (type==1)

{

++n;

a[n].d[0]=getint();a[n].d[1]=getint();

a[n].minn[0]=a[n].maxn[0]=a[n].d[0];

a[n].minn[1]=a[n].maxn[1]=a[n].d[1];

KDTree::Insert(n);

}

else

{

x=getint();y=getint();ans=INF;

KDTree::query(root,x,y);

printf("%d\n",ans);

}

}

return 0;

}