K-D tree

K-D树是在k维欧几里得空间中组织点的数据结构。在算法竞赛中，K-D树往往用于在二维平面内的信息检索。具体应用如：多维键值搜索（范围搜索及最邻近搜索）下面我们针对二维平面的KD-Tree进行讲解。

定义

K-D Tree是一颗存储k维信息的二叉树，代表对数据集合的划分，每个结点都对应着一个k维超矩形区域。

①在一维数据情况下，KD-Tree是一颗二叉搜索树。

②二维情况下，我们轮流按照x维和y维对数据进行划分。这是最简单的一种划分方式，使得每次划分都将平面分成不相交的两部分。

建树

对于KD-Tree每个结点的存储信息如下：

```
struct KDTree{
    int ch[2],id;  //左右儿子和当前结点的编号
    Point p,r1,r2;
    //结点表示的点，子树赋改的矩形的左下角，右上角
}
```

每次新建结点是根据当前划分维度选中位数进行划分，注意$nth_element$函数的使用，是线性时间选择中位数，并将比他小的元素排序到左边，比他大的元素排序到右边的。

```
int build(int l,int r,int d) {  //KD-Tree建树
    if (l>r) return 0;
    del=d;  //改变划分维度
    int mid=(l+r)>>1,at=++ncnt;
    nth_element(ps+l,ps+mid,ps+r+1,cmp); //O(n)找到第mid大元素，将比他小的放左边，比他大的放右边
    T[at]=Tree(ps[mid],mid);
    T[at].ch[0]=build(l,mid-1,d^1),T[at].ch[1]=build(mid+1,r,d^1);
    pushup(at);return at;
}
```

更新

对于KD-Tree每个结点表示的矩形，需要由左右儿子向上更新

```
void pushup(int rt) {  //更新操作
    T[rt].r1.x=min(min(T[ls].r1.x,T[rs].r1.x),T[rt].r1.x);
    T[rt].r1.y=min(min(T[ls].r1.y,T[rs].r1.y),T[rt].r1.y);
    T[rt].r2.x=max(max(T[ls].r2.x,T[rs].r2.x),T[rt].r2.x);
    T[rt].r2.y=max(max(T[ls].r2.y,T[rs].r2.y),T[rt].r2.y);
}
```

插入

插入是根据当前层比较的维度，较小的插左子树，较大的插右子树，比较简单。

```
void modify(int rt,Point p){
    if(!rt){T[rt]=Tree(P,++ncnt);return;}
    int d=cmp(p,T[rt].p)^1;
    del^=1;
    modify(T[rt].ch[d],p);
    pushup(T[rt]);
}
```

查询

最近点查询

1.设定答案ans为初始值∞

2.将点P从根节点开始，先用根节点代表的点更新答案。由于根节点的左右儿子各表示一个矩形区域，而两个区域都有可能存在距离P最近的点，我们优先选择距离点P最近的矩形递归查询。

3.以P为圆心，ans为半径画圆，如果与之前未递归的矩形相交，则递归下去，否则不可能由更优解。

事实上剪枝+搜素，单次查询均摊复杂度是$O(logn)$,最坏$O(\sqrt{n})$

```
void query(int rt,Point p) {  //查询操作
    if (!rt) return;
    node st=node(dis(T[rt].p,p),T[rt].p.id);
    if (st<q.top()) q.pop(),q.push(st);
    double dis[2]={T[ls].dis(p),T[rs].dis(p)};
    int nxt=dis[0]<dis[1];
    query(T[rt].ch[nxt],p);  //优先选择两个儿子中矩形距离小的那个
    if (node(dis[nxt^1],T[T[rt].ch[nxt^1]].id)<q.top())
query(T[rt].ch[nxt^1],p); //如果另一个儿子有可能比当前结果小，就递归下去
}
```

K远点

在寻找最近点时，我们维护一个大根堆，一开始放k个$-\infty$，然后维护查询结果即可。

例题

luoguP4357 [CQOI2016]K 远点对

题意：给定k个点，求第k远点对的距离平方。

将所有点插进KD-Tree中维护小顶堆即可。

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int N=1e5+7;

const double inf=1e16;
struct Point {  //二维平面的点
    int x,y,id;
    Point (int x=0,int y=0):x(x),y(y) {}
}ps[N];

bool del;  //比较x为0，比较y为1
bool cmp(Point p1,Point p2) {
    if (!del) return (p1.x<p2.x||(p1.x==p2.x&&p1.y<p2.y));
    return (p1.y<p2.y||(p1.y==p2.y&&p1.x<p2.x));
}
int dis(Point p1,Point p2) { //计算两点距离
    return (p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y);
}
#define ls T[rt].ch[0]
#define rs T[rt].ch[1]
struct Tree { //KD-Tree
    int ch[2],id;Point p,r1,r2; //左右儿子和当前结点编号，当前点和表示平面的左下右上角
    Tree(Point p=Point(),int id=0):p(p),r1(p),r2(p),id(id) {}
    double dis(Point p) {
        if (!id) return -inf;
        return
max(max(::dis(p,r1),::dis(p,r2)),max(::dis(p,Point(r1.x,r2.y)),::dis(p,Point(r2.
x,r1.y))));
    }
}T[N];
```

```
void pushup(int rt) {  //更新操作
    T[rt].r1.x=min(min(T[ls].r1.x,T[rs].r1.x),T[rt].r1.x);
    T[rt].r1.y=min(min(T[ls].r1.y,T[rs].r1.y),T[rt].r1.y);
    T[rt].r2.x=max(max(T[ls].r2.x,T[rs].r2.x),T[rt].r2.x);
    T[rt].r2.y=max(max(T[ls].r2.y,T[rs].r2.y),T[rt].r2.y);
}
int ncnt=0;
void init() {  //初始化平面
    T[0].r1=Point(0x3f3f3f3f,0x3f3f3f3f),T[0].r2=Point(-0x3f3f3f3f,-0x3f3f3f3f);
}
int build(int l,int r,int d) {  //KD-Tree建树
    if(l>r) return 0;
    del=d;  //改变划分维度
    int mid=(l+r)>>1,at=++ncnt;
    nth_element(ps+l,ps+mid,ps+r+1,cmp); //O(n)找到第mid大元素，将比他小的放左边，比他大的放右边
    T[at]=Tree(ps[mid],mid);
    T[at].ch[0]=build(l,mid-1,d^1),T[at].ch[1]=build(mid+1,r,d^1);
    pushup(at);return at;
}
struct node {  //堆中的结点
    int dis;int id;
    node(int dis=0,int id=0):dis(dis),id(id) {}
    bool operator < (node b) const {
        return dis>b.dis||(dis==b.dis&&id<b.id);
    }
};
priority_queue<node>q; //大根堆
priority_queue<int>ans;

void query(int rt,Point p) {  //查询操作
    if(!rt) return;
    node st=node(dis(T[rt].p,p),T[rt].p.id);
    if(st<q.top()) q.pop(),q.push(st);
    double dis[2]={T[ls].dis(p),T[rs].dis(p)};
    int nxt=dis[0]<dis[1];
    query(T[rt].ch[nxt],p);  //优先选择两个儿子中矩形距离小的那个
    if(node(dis[nxt^1],T[T[rt].ch[nxt^1]].id)<q.top()) query(T[rt].ch[nxt^1],p);
//如果另一个儿子有可能比当前结果小，就递归下去
}
signed main() {
    init();
    int n,k;
    scanf("%lld%lld",&n,&k);
    k*=2;
    for(int i=1;i<=n;i++) scanf("%lld%lld",&ps[i].x,&ps[i].y),ps[i].id=i;
    build(1,n,0);
    while (!q.empty()) q.pop();  //初始化队列
    for(int j=1;j<=k;j++) q.push(node(-inf)); //设定答案初始值
    for(int i=1;i<=n;i++) query(1,ps[i]);
    cout<<q.top().dis;
    return 0;
}
```

luogu P2093 [国家集训队]JZPFAR

题意：求平面内距离给定点第k远的点。

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+7;

const double inf=1e16;
struct Point {  //二维平面的点
    int x,y,id;
    Point (int x=0,int y=0):x(x),y(y) {}
}ps[N];

bool del;  //比较x为0，比较y为1
bool cmp(Point p1,Point p2) {
    if (!del) return (p1.x<p2.x||(p1.x==p2.x&&p1.y<p2.y));
    return (p1.y<p2.y||(p1.y==p2.y&&p1.x<p2.x));
}
double dis(Point p1,Point p2) { //计算两点距离
    return (double)(p1.x-p2.x)*(p1.x-p2.x)+(double)(p1.y-p2.y)*(p1.y-p2.y);
}
#define ls T[rt].ch[0]
#define rs T[rt].ch[1]
struct Tree { //KD-Tree
    int ch[2],id;Point p,r1,r2; //左右儿子和当前结点编号，当前点和表示平面的左下右上角
    Tree(Point p=Point(),int id=0):p(p),r1(p),r2(p),id(id) {}
    double dis(Point p) {
        if (!id) return -inf;
        return
max(max(::dis(p,r1),::dis(p,r2)),max(::dis(p,Point(r1.x,r2.y)),::dis(p,Point(r2.
x,r1.y))));
    }
}T[N];
void pushup(int rt) {  //更新操作
    T[rt].r1.x=min(min(T[ls].r1.x,T[rs].r1.x),T[rt].r1.x);
    T[rt].r1.y=min(min(T[ls].r1.y,T[rs].r1.y),T[rt].r1.y);
    T[rt].r2.x=max(max(T[ls].r2.x,T[rs].r2.x),T[rt].r2.x);
    T[rt].r2.y=max(max(T[ls].r2.y,T[rs].r2.y),T[rt].r2.y);
}
int ncnt=0;
void init() {  //初始化平面
    T[0].r1=Point(0x3f3f3f3f,0x3f3f3f3f),T[0].r2=Point(-0x3f3f3f3f,-0x3f3f3f3f);
}
int build(int l,int r,int d) {  //KD-Tree建树
    if (l>r) return 0;
    del=d;  //改变划分维度
    int mid=(l+r)>>1,at=++ncnt;
    nth_element(ps+l,ps+mid,ps+r+1,cmp); //O(n)找到第mid大元素，将比他小的放左边，比他
大的放右边
    T[at]=Tree(ps[mid],mid);
    T[at].ch[0]=build(l,mid-1,d^1),T[at].ch[1]=build(mid+1,r,d^1);
    pushup(at);return at;
}
struct node {  //堆中的结点
    double dis;int id;
    node(double dis=0,int id=0):dis(dis),id(id) {}
```

```
    bool operator < (node b) const {
        return dis>b.dis||(dis==b.dis&&id<b.id);
    }
};
priority_queue<node> q; //大根堆
void query(int rt,Point p) {  //查询操作
    if (!rt) return;
    node st=node(dis(T[rt].p,p),T[rt].p.id);
    if (st<q.top()) q.pop(),q.push(st);
    double dis[2]={T[ls].dis(p),T[rs].dis(p)};
    int nxt=dis[0]<dis[1];
    query(T[rt].ch[nxt],p);  //优先选择两个儿子中矩形距离小的那个
    if (node(dis[nxt^1],T[T[rt].ch[nxt^1]].id)<q.top())
query(T[rt].ch[nxt^1],p); //如果另一个儿子有可能比当前结果小，就递归下去
}
int main() {
    init();
    int n,m;scanf("%d",&n);
    for (int i=1;i<=n;i++) scanf("%d%d",&ps[i].x,&ps[i].y),ps[i].id=i;
    build(1,n,0);
    scanf("%d",&m);
    int x,y,k;
    while (m--) {
        scanf("%d%d%d",&x,&y,&k);
        while (!q.empty()) q.pop();  //初始化队列
        for (int i=1;i<=k;i++) q.push(node(-inf)); //设定答案初始值
        query(1,Point(x,y));  //从根节点出发查询答案
        printf("%d\n",q.top().id);
    }
    return 0;
}
```

P4148 简单题

在二维坐标平面上支持：

- 插入某个点
- 查询一个矩形内的所有点权值之和

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+7;
double alpha=0.75;

int n,rt,tot=0,top=0,cnt=0;

struct P{   //存储点的信息
    int x[2],val;
}p[N];

#define lc(x) t[x].lc
#define rc(x) t[x].rc

struct Tree{   //KD-Tree存储矩形信息
```

```
    int lc,rc,sz,sum;
    int mx[2],mi[2];
    P pt;
}t[N];

int WD,rub[N];

bool cmp(P a,P b){return a.x[WD]<b.x[WD];}

int newnode(){  //获取结点编号
    if(top) return rub[top--];
    else return ++tot;
}

void update(int x){   //更新结点信息
    t[x].sz=t[lc(x)].sz+t[rc(x)].sz+1;
    t[x].sum=t[lc(x)].sum+t[rc(x)].sum+t[x].pt.val;
    for(int i=0;i<=1;++i){
        t[x].mx[i]=t[x].mi[i]=t[x].pt.x[i];
        if(lc(x)){
            t[x].mx[i]=max(t[x].mx[i],t[lc(x)].mx[i]);
            t[x].mi[i]=min(t[x].mi[i],t[lc(x)].mi[i]);
        }
        if(rc(x)){
            t[x].mx[i]=max(t[x].mx[i],t[rc(x)].mx[i]);
            t[x].mi[i]=min(t[x].mi[i],t[rc(x)].mi[i]);
        }
    }
}

bool bad(int x){  //是否不平衡
    return t[x].sz*alpha<t[lc(x)].sz||t[x].sz*alpha<t[rc(x)].sz;
}

void work(int x){  //获取中序遍历
    if(lc(x)) work(lc(x));
    p[++cnt]=t[x].pt;
    rub[++top]=x;
    if(rc(x)) work(rc(x));
}

int build(int l,int r,int wd){
    if(l>r) return 0;
    int mid=(l+r)>>1,x=newnode();
    WD=wd;
    nth_element(p+l,p+mid,p+r+1,cmp);  //线性查找中位数
    t[x].pt=p[mid];
    t[x].lc=build(l,mid-1,wd^1);
    t[x].rc=build(mid+1,r,wd^1);
    update(x);
    return x;
}

void rebuild(int &x){  //暴力重构
    cnt=0;
    work(x);   //拍扁
    x=build(1,cnt,0);
}
```

```
void ins(int &x,P tmp,int wd){
    if(!x){  //新建节点
        x=newnode();
        t[x].lc=t[x].rc=0;
        t[x].pt=tmp;
        update(x);
        return;
    }
    if(tmp.x[wd]<=t[x].pt.x[wd]) ins(lc(x),tmp,wd^1);
    else ins(rc(x),tmp,wd^1);  //判断插入左子树还是右子树
    update(x);
    if(bad(x)) rebuild(x);
}

bool in(int nx1,int nx2,int ny1,int ny2,int x1,int y1,int x2,int y2){ //是否完全在矩形里
    if(nx1>=x1&&nx2<=x2&&ny1>=y1&&ny2<=y2) return 1;
    return 0;
}

bool out(int nx1,int nx2,int ny1,int ny2,int x1,int y1,int x2,int y2){ //是否不在矩形里
    if(x1>nx2||x2<nx1||y1>ny2||y2<ny1) return 1;
    return 0;
}

int query(int p,int x1,int y1,int x2,int y2){  //查询操作
    if(!p) return 0;
    if(out(t[p].mi[0],t[p].mx[0],t[p].mi[1],t[p].mx[1],x1,y1,x2,y2)) return 0;
    if(in(t[p].mi[0],t[p].mx[0],t[p].mi[1],t[p].mx[1],x1,y1,x2,y2)) return t[p].sum;
    int res=0;
    if(in(t[p].pt.x[0],t[p].pt.x[0],t[p].pt.x[1],t[p].pt.x[1],x1,y1,x2,y2)) res+=t[p].pt.val; //当前点在矩阵内
    return query(t[p].lc,x1,y1,x2,y2)+query(t[p].rc,x1,y1,x2,y2)+res;
}

signed main(){
    scanf("%d",&n);
    int op,a,x,y,x2,y2,lst=0;
    while(1){
        scanf("%d",&op);
        if(op==1){
            scanf("%d%d%d",&x,&y,&a);
            x^=lst,y^=lst,a^=lst;
            ins(rt,(P){x,y,a},0);
        }else if(op==2){
            scanf("%d%d%d%d",&x,&y,&x2,&y2);
            x^=lst;y^=lst;x2^=lst;y2^=lst;
            printf("%d\n",lst=query(rt,x,y,x2,y2));
        }else break;
    }
}
```

[P4357 CQOI2016]K远点对

已知平面内n个点的坐标，求欧式距离下的第k远点对。

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+7;
double alpha=0.75;
typedef long long ll;

priority_queue<ll,vector<ll>,greater<ll>>q;
int n,k,rt,tot=0,top=0,cnt=0;

struct P{   //存储点的信息
    int x[2],val;
}p[N];

#define lc(x) t[x].lc
#define rc(x) t[x].rc

struct Tree{   //KD-Tree存储矩形信息
    int lc,rc,sz,sum;
    int mx[2],mi[2];
    P pt;
}t[N];

int WD,rub[N];

bool cmp(P a,P b){return a.x[WD]<b.x[WD];}

int newnode(){  //获取结点编号
    if(top) return rub[top--];
    else return ++tot;
}

void update(int x){   //更新结点信息
    t[x].sz=t[lc(x)].sz+t[rc(x)].sz+1;
    t[x].sum=t[lc(x)].sum+t[rc(x)].sum+t[x].pt.val;
    for(int i=0;i<=1;++i){
        t[x].mx[i]=t[x].mi[i]=t[x].pt.x[i];
        if(lc(x)){
            t[x].mx[i]=max(t[x].mx[i],t[lc(x)].mx[i]);
            t[x].mi[i]=min(t[x].mi[i],t[lc(x)].mi[i]);
        }
        if(rc(x)){
            t[x].mx[i]=max(t[x].mx[i],t[rc(x)].mx[i]);
            t[x].mi[i]=min(t[x].mi[i],t[rc(x)].mi[i]);
        }
    }
}

bool bad(int x){  //是否不平衡
    return t[x].sz*alpha<t[lc(x)].sz||t[x].sz*alpha<t[rc(x)].sz;
}

void work(int x){  //获取中序遍历
    if(lc(x)) work(lc(x));
```

```
    p[++cnt]=t[x].pt;
    rub[++top]=x;
    if(rc(x)) work(rc(x));
}

int build(int l,int r,int wd){
    if(l>r) return 0;
    int mid=(l+r)>>1,x=newnode();
    WD=wd;
    nth_element(p+l,p+mid,p+r+1,cmp);  //线性查找中位数
    t[x].pt=p[mid];
    t[x].lc=build(l,mid-1,wd^1);
    t[x].rc=build(mid+1,r,wd^1);
    update(x);
    return x;
}

void rebuild(int &x){  //暴力重构
    cnt=0;
    work(x);   //拍扁
    x=build(1,cnt,0);
}

void ins(int &x,P tmp,int wd){
    if(!x){  //新建节点
        x=newnode();
        t[x].lc=t[x].rc=0;
        t[x].pt=tmp;
        update(x);
        return;
    }
    if(tmp.x[wd]<=t[x].pt.x[wd]) ins(lc(x),tmp,wd^1);
    else ins(rc(x),tmp,wd^1);  //判断插入左子树还是右子树
    update(x);
    if(bad(x)) rebuild(x);
}

inline ll poww(int x){return (ll)x*x;}
//计算可能的最大距离，仅用来剪枝，并不精确
ll getdis(int k,int x,int y){
    ll res=0ll;
    res+=max(poww(t[k].mx[0]-x),poww(t[k].mi[0]-x));
    res+=max(poww(t[k].mx[1]-y),poww(t[k].mi[1]-y));
    return res;
}

void work(int k,int x,int y){
    ll dis=0ll,dl=0ll,dr=0ll;
    dis=poww(t[k].pt.x[0]-x)+poww(t[k].pt.x[1]-y);  //计算当前结点到目标结点的距离
    if(lc(k)) dl=getdis(lc(k),x,y);
    if(rc(k)) dr=getdis(rc(k),x,y);
    if(dis>q.top()) q.pop(),q.push(dis);
    if(dl>dr){  //剪枝，先遍历大的
        if(dl>q.top()) work(lc(k),x,y);
        if(dr>q.top()) work(rc(k),x,y);
    }else{
        if(dr>q.top()) work(rc(k),x,y);
        if(dl>q.top()) work(lc(k),x,y);
```

```
    }
}

signed main(){
    scanf("%d%d",&n,&k);
    k<<=1;
    for(int i=1;i<=k;++i) q.push(0);
    for(int i=1;i<=n;++i) scanf("%d%d",&p[i].x[0],&p[i].x[1]);
    rt=build(1,n,0);
    for(int i=1;i<=n;++i) work(rt,p[i].x[0],p[i].x[1]);
    printf("%lld\n",q.top());
    return 0;
}
```

SDOI2010]捉迷藏

给n个点，选出一个点使得距离该点最远的点到它的距离和距离该点最近的点到它的距离之差最小，输出最小值。（曼哈顿距离）

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=100010;
const ll INF=0x7fffffffff;
int n,topp=0,nodetot=0,WD,rt;
struct point {
    int x[2];
} p[N];
struct tree {
    int lc,rc,siz,maxn[2],minn[2];
    point pt;
} t[N];
int rub[N];
int newnode() {
    if(topp)return rub[topp--];
    return ++nodetot;
}
bool cmp(point a,point b) {
    return a.x[WD]<b.x[WD];
}
void update(int k) {
    t[k].siz=t[t[k].lc].siz+t[t[k].rc].siz+1;
    for(int i=0; i<=1; i++) {
        t[k].maxn[i]=t[k].minn[i]=t[k].pt.x[i];
        if(t[k].lc) {
            t[k].maxn[i]=max(t[k].maxn[i],t[t[k].lc].maxn[i]);
            t[k].minn[i]=min(t[k].minn[i],t[t[k].lc].minn[i]);
        }
        if(t[k].rc) {
            t[k].maxn[i]=max(t[k].maxn[i],t[t[k].rc].maxn[i]);
            t[k].minn[i]=min(t[k].minn[i],t[t[k].rc].minn[i]);
        }
    }
}
int build(int l,int r,int wd) {
```

```
    if(l>r)return 0;
    int mid=(l+r)>>1,k=newnode();
    WD=wd;
    nth_element(p+l,p+mid,p+r+1,cmp);
    t[k].pt=p[mid];
    t[k].lc=build(l,mid-1,wd^1);
    t[k].rc=build(mid+1,r,wd^1);
    update(k);
    return k;
}
ll maxd,mind;
ll getmax(int k,int x,int y) {
    ll res=0;
    res+=max(abs(t[k].maxn[0]-x),abs(t[k].minn[0]-x));
    res+=max(abs(t[k].maxn[1]-y),abs(t[k].minn[1]-y));
    return res;
}
ll getmin(int k,int x,int y) {
    ll res=0ll;
    if(x<t[k].minn[0])res+=t[k].minn[0]-x;
    else if(x>t[k].maxn[0])res+=x-t[k].maxn[0];
    if(y<t[k].minn[1])res+=t[k].minn[1]-y;
    else if(y>t[k].maxn[1])res+=y-t[k].maxn[1];
    return res;
}
ll getdis(int k,int x,int y) { //曼哈顿距离
    return abs(t[k].pt.x[0]-x)+abs(t[k].pt.x[1]-y);
}
void query_max(int k,int x,int y) {
    if(!k)return ;
    ll dis=0ll,dl=0ll,dr=0ll;
    dis=getdis(k,x,y);
    maxd=max(maxd,dis);
    if(t[k].lc)dl=getmax(t[k].lc,x,y);
    if(t[k].rc)dr=getmax(t[k].rc,x,y);
    if(dl>dr) {
        if(dl>maxd)query_max(t[k].lc,x,y);
        if(dr>maxd)query_max(t[k].rc,x,y);
    } else {
        if(dr>maxd)query_max(t[k].rc,x,y);
        if(dl>maxd)query_max(t[k].lc,x,y);
    }
}
void query_min(int k,int x,int y) {
    if(!k)return ;
    ll dis,dl=INF,dr=INF;
    dis=getdis(k,x,y);
    if(dis)mind=min(mind,dis);
    if(t[k].lc)dl=getmin(t[k].lc,x,y);
    if(t[k].rc)dr=getmin(t[k].rc,x,y);
    if(dl<dr) {
        if(dl<mind)query_min(t[k].lc,x,y);
        if(dr<mind)query_min(t[k].rc,x,y);
    } else {
        if(dr<mind)query_min(t[k].rc,x,y);
        if(dl<mind)query_min(t[k].lc,x,y);
    }
}
```

```
int main() {
    scanf("%d",&n);
    for(int i=1; i<=n; i++)scanf("%d %d",&p[i].x[0],&p[i].x[1]);
    rt=build(1,n,0);
    ll ans=INF;
    for(int i=1; i<=n; i++) {
        maxd=-INF,mind=INF;
        query_max(rt,p[i].x[0],p[i].x[1]);
        query_min(rt,p[i].x[0],p[i].x[1]);
        ans=min(ans,(maxd-mind));
    }
    printf("%d",ans);
    return 0;
}
```

CH弱省胡策R2]TATT

四维偏序（给定四维空间的点，求最长坐标升序序列长度）

```
#include<bits/stdc++.h>
using namespace std;
const int N=500010;
const int INF=1e9+7;
const double alpha=0.725;
int n,topp=0,nodetot=0,rt,tot=0;
int rub[N];
int WD;
struct point {
    int x[4],val;
} p[N],o[N];
struct tree {
    int lc,rc,siz,mx,d;
    int maxn[4],minn[4];
    point pt;
} t[N];
bool cmp(point a,point b) {
    return a.x[WD]<b.x[WD];
}
int newnode() {
    if(topp)return rub[topp--];
    return ++nodetot;
}
void update(int k) {
    t[k].mx=max(t[k].pt.val,max(t[t[k].lc].mx,t[t[k].rc].mx));
    t[k].siz=t[t[k].lc].siz+t[t[k].rc].siz+1;
    for(int i=1; i<=3; i++) {
        t[k].maxn[i]=t[k].minn[i]=t[k].pt.x[i];
        if(t[k].lc) {
            t[k].maxn[i]=max(t[k].maxn[i],t[t[k].lc].maxn[i]);
            t[k].minn[i]=min(t[k].minn[i],t[t[k].lc].minn[i]);
        }
        if(t[k].rc) {
            t[k].maxn[i]=max(t[k].maxn[i],t[t[k].rc].maxn[i]);
            t[k].minn[i]=min(t[k].minn[i],t[t[k].rc].minn[i]);
        }
```

```
    }
}
int build(int l,int r,int wd) {
    if(l>r)return 0;
    int mid=(l+r)>>1,k=newnode();
    WD=wd;
    nth_element(o+l,o+mid,o+r+1,cmp);
    t[k].pt=o[mid];
    t[k].d=wd;
    t[k].lc=build(l,mid-1,(wd+1)%3+1);
    t[k].rc=build(mid+1,r,(wd+1)%3+1);
    update(k);
    return k;
}
bool cmp1(point a,point b) { //先排序第一维
    for(int i=0; i<=3; i++) {
        if(a.x[i]<b.x[i])return 1;
        if(a.x[i]>b.x[i])return 0;
    }
    return 0;
}
int ans=0;
int check(point a,point b) {
    for(int i=1; i<=3; i++)if(a.x[i]>b.x[i])return 0;
    return 1;
}
int inandout(int a,point b) {
    if(!a)return 0;
    int cnt=0;
    for(int i=1; i<=3; i++)
        if(t[a].maxn[i]<=b.x[i])cnt++;
    if(cnt==3)return cnt;
    cnt=1;
    for(int i=1; i<=3; i++) {
        if(t[a].minn[i]>b.x[i]) { //只要有一个最小值大于目标点，就一定不满足
            cnt=0;
            break;
        }
    }
    return cnt;
}
void query(int k,point tmp) {
    if(check(t[k].pt,tmp))ans=max(ans,t[k].pt.val);//当前处理到的点是否满足偏序
    int dl=inandout(t[k].lc,tmp),dr=inandout(t[k].rc,tmp);
    if(dl==3)ans=max(ans,t[t[k].lc].mx);//左子树的矩形完全满足偏序
    else if(dl&&ans<t[t[k].lc].mx)query(t[k].lc,tmp);//部分满足
    if(dr==3)ans=max(ans,t[t[k].rc].mx);
    else if(dr&&ans<t[t[k].rc].mx)query(t[k].rc,tmp);
}
bool bad(int k) {
    if(t[k].siz*alpha<t[t[k].lc].siz||t[k].siz*alpha<t[t[k].rc].siz)return 1;
    return 0;
}
void work(int k) {
    if(t[k].lc)work(t[k].lc);
    o[++tot]=t[k].pt;
    rub[++topp]=k;
    if(t[k].rc)work(t[k].rc);
```

```
}
void rebuild(int &k) {
    tot=0;
    work(k);
    k=build(1,tot,t[k].d);//从当前的维度开始建，优化时间
}
void ins(int &k,point tmp,int wd) {
    if(!k) {
        k=newnode();
        t[k].lc=t[k].rc=0;
        t[k].d=wd;//这里特殊记录一下维度
        t[k].pt=tmp;
        update(k);
        return ;
    }
    if(t[k].pt.x[wd]<=tmp.x[wd])ins(t[k].lc,tmp,(wd+1)%3+1);
    else ins(t[k].rc,tmp,(wd+1)%3+1);
    update(k);
    if(bad(k))rebuild(k);
}
int main() {
    int n;
    scanf("%d",&n);
    for(int i=1; i<=n; i++)scanf("%d %d %d
%d",&p[i].x[0],&p[i].x[1],&p[i].x[2],&p[i].x[3]),p[i].val=1;
    sort(p+1,p+n+1,cmp1);
    int output=0;
    for(int i=1; i<=n; i++) {
        ans=0;
        query(rt,p[i]);
        p[i].val+=ans;
        output=max(output,p[i].val);
        ins(rt,p[i],1);
    }
    printf("%d\n",output);
    return 0;
}
/*
4
2 3 33 2333
2 3 33 2333
2 3 33 2333
2 3 33 2333
*/
```

[SDOI2012]最近最远点对

求平面中两个点的最远距离和最近距离。（欧几里得距离）

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=1e5+5;
int n,now,ls[maxn],rs[maxn],rt;
double mxx[maxn],mxy[maxn],mnx[maxn],mny[maxn],ans1=9e18,ans2;
struct node{
```

```
    double x,y;
}a[maxn];
void pushup(int u){
    mxx[u]=max(mxx[u],a[now].x);mxy[u]=max(mxy[u],a[now].y);
    mnx[u]=min(mnx[u],a[now].x);mny[u]=min(mny[u],a[now].y);
}
double sqr(double x){
    return x*x;
}
double dis(int u){
    return sqr(a[now].x-a[u].x)+sqr(a[now].y-a[u].y);
}
double mindis(int u){//子树u中最小距离
    return sqr(max(a[now].x-mxx[u],0.0)+max(mnx[u]-
a[now].x,0.0))+sqr(max(a[now].y-mxy[u],0.0)+max(mny[u]-a[now].y,0.0));
}
double maxdis(int u){//子树u中最小距离
    return max(sqr(a[now].x-mxx[u]),sqr(mnx[u]-a[now].x))+max(sqr(a[now].y-
mxy[u]),sqr(mny[u]-a[now].y));
}
void ins(int &u,bool op){
    if(!u)return u=now,void();
    if(!op)ins(a[now].x<=a[u].x?ls[u]:rs[u],1);
    else ins(a[now].y<=a[u].y?ls[u]:rs[u],0);
    pushup(u);
}
void askmin(int u){
    if(!u)return;
    if(u!=now)ans1=min(ans1,dis(u));
    double l=mindis(ls[u]),r=mindis(rs[u]);
    if(l<r){//剪枝：如果比答案大就不搜了
        if(l<ans1)askmin(ls[u]);
        if(r<ans1)askmin(rs[u]);
    }else{
        if(r<ans1)askmin(rs[u]);
        if(l<ans1)askmin(ls[u]);
    }
}
void askmax(int u){
    if(!u)return;
    ans2=max(ans2,dis(u));
    double l=maxdis(ls[u]),r=maxdis(rs[u]);
    if(l>r){
        if(l>ans2)askmax(ls[u]);
        if(r>ans2)askmax(rs[u]);
    }else{
        if(r>ans2)askmax(rs[u]);
        if(l>ans2)askmax(ls[u]);
    }
}
int main(){
    srand(20201213);//今天
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
        scanf("%lf%lf",&a[i].x,&a[i].y);
    random_shuffle(a+1,a+n+1);
    for(int i=1;i<=n;i++){
        mxx[i]=mnx[i]=a[i].x;
```

```
        mxy[i]=mny[i]=a[i].y;
        now++;askmin(rt);askmax(rt);ins(rt,0);
    }printf("%.2lf %.2lf\n",sqrt(ans1),sqrt(ans2));
    return 0;
}
```

参考资料

https://blog.sengxian.com/algorithms/k-dimensional-tree

https://www.jianshu.com/p/4fd7cad27906