KD_Tree

算法简介

一种特殊的二叉树，主要用于多维空间关键数据的搜索。

空间复杂度 $O(n)$□单次插入时间复杂度 $O(\log n)$□查询时间复杂度 $O\left(k\sqrt[1-\frac 1k]n\right)$□其中 k 表示空间维数。

算法实现

为方便理解，这里仅讲解 2D_Tree□高维 KD_Tree 可以类推。实际上高维 KD_Tree 时间复杂度难以承受，算法竞赛中通常只涉及 2D_Tree□

先考虑建树过程。

二维空间的点无法直接比较大小，但如果将某个维度作为主要关键字，另一个维度作为次要关键字就可以使得比较大小成为可能。

每层都选用一个维度，对结点进行排序，取中间结点的该维度数值作为分割面，将该结点左边结点加入左子树，该结点右边结点加入右子树。

algorithm 库里有个叫 nth_element 的神奇函数，可以 $O(n)$ 完成上述操作。

不断重复上述过程，便可以完成建树，而且可以使得该树高度平衡，时间复杂度 $O(n\log n)$□

建树过程实际上将整个二维空间分割成了若干部分。为了方便后面查询操作的剪枝，需要维护每个结点的子树的最小覆盖矩阵。

为使空间分割尽量均匀，需要选择合适的关键字。

比较优秀的关键字选择方法为求每个维度方差，选取方差大的维度作为主要关键字。

然而上述方法代码复杂，算法竞赛一般考虑轮换的方法选取主要关键字。

接下来是插入操作，插入操作会破坏原本树的平衡性，这个问题可以用替罪羊树的重构解决。

最后是查询操作，查询其实就是个暴力查询，但可以利用最小覆盖矩阵的剪枝控制时间复杂度，详细见例题。

代码模板

洛谷p4169

模板题

题意

二维空间，一开始 n 个点□ m 个操作。

操作一：加入点 (x,y)。

操作二：询问当前点集中到给定点 (x,y) 的最小哈密顿距离。

题解

建树、插入操作不再赘述。关于查询操作，将当前查询结果 ans 设为全局变量，从根结点开始遍历 KD_Tree。假设当前访问结点为 pos。

首先用 pos 到给定点 (x,y) 的哈密顿距离更新 ans。

计算给定点 (x,y) 到 pos 的两个儿子结点的最小覆盖矩阵的最小哈密顿距离，记为 d_1、d_2。

优先遍历 d_i 较小的结点。若 $d_i\gt ans$ 立刻剪枝。

```
const int MAXN=6e5+5,inf=1e9;
const double alpha=0.75;
struct Point{
    int x,y;
    Point(int x=0,int y=0):x(x),y(y){}
    int get_dis(const Point &P){
        return abs(x-P.x)+abs(y-P.y);
    }
    void get_min(const Point &a,const Point &b){
        x=min(x,min(a.x,b.x));
        y=min(y,min(a.y,b.y));
    }
    void get_max(const Point &a,const Point &b){
        x=max(x,max(a.x,b.x));
        y=max(y,max(a.y,b.y));
    }
};
struct Node{
    int ch[2],cnt;
    Point p,r1,r2;
    void build(Point P){
        p=r1=r2=P;
        ch[0]=ch[1]=0;
        cnt=1;
    }
    int get_value(Point P){
        return max(r1.x-P.x,0)+max(P.x-r2.x,0)+max(r1.y-P.y,0)+max(P.y-
r2.y,0);
    }
}node[MAXN];
bool isbad(int pos){return
alpha*node[pos].cnt<max(node[node[pos].ch[0]].cnt,node[node[pos].ch[1]].cnt
)?true:false;}
void maintain(int pos){
```

```
node[pos].r1.get_min(node[node[pos].ch[0]].r1,node[node[pos].ch[1]].r1);
node[pos].r2.get_max(node[node[pos].ch[0]].r2,node[node[pos].ch[1]].r2);
    node[pos].cnt=node[node[pos].ch[0]].cnt+node[node[pos].ch[1]].cnt+1;
}
int pool[MAXN],pos1,pos2,root,dimension;
Point temp[MAXN];
bool cmp(const Point &p1,const Point &p2){
    if(!dimension)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);
}
void Init(int n){
    node[0].r1=Point(inf,inf);
    node[0].r2=Point(-inf,-inf);
    for(int i=n;i>=1;i--)
    pool[++pos1]=i;
}
void build(int &pos,int lef,int rig,bool d){
    if(lef>rig) return pos=0,void();
    pos=pool[pos1--];
    int mid=lef+rig>>1;
    dimension=d;
    nth_element(temp+lef,temp+mid,temp+rig+1,cmp);
    node[pos].p=node[pos].r1=node[pos].r2=temp[mid];
    build(node[pos].ch[0],lef,mid-1,!d);
    build(node[pos].ch[1],mid+1,rig,!d);
    maintain(pos);
}
void build(int n){build(root,1,n,false);}
void dfs(int pos){
    if(!pos)return;
    dfs(node[pos].ch[0]);
    pool[++pos1]=pos,temp[++pos2]=node[pos].p;
    dfs(node[pos].ch[1]);
}
void rebuild(int &pos,bool d){
    pos2=0;
    dfs(pos);
    build(pos,1,pos2,d);
}
void check(int &pos,Point x,bool d){
    if(pos){
        if(isbad(pos)){
            rebuild(pos,d);
            return;
        }
        dimension=d;
        if(cmp(node[pos].p,x))
        check(node[pos].ch[1],x,!d);
        else
        check(node[pos].ch[0],x,!d);
    }
```

```
}
void Insert(int &pos,Point x,bool d){
    if(!pos){
        pos=pool[pos1--];
        node[pos].build(x);
        return;
    }
    node[pos].cnt++;
    dimension=d;
    if(cmp(node[pos].p,x))Insert(node[pos].ch[1],x,!d);
    else Insert(node[pos].ch[0],x,!d);
    maintain(pos);
}
void Insert(Point x){
    Insert(root,x,false);
    check(root,x,false);
}
int ans;
void query(int pos,Point x){
    if(!pos)
    return;
    int min_ans[2],dir;
    ans=min(ans,node[pos].p.get_dis(x));
    min_ans[0]=node[pos].ch[0]?node[node[pos].ch[0]].get_value(x):inf;
    min_ans[1]=node[pos].ch[1]?node[node[pos].ch[1]].get_value(x):inf;
    dir=min_ans[0]<min_ans[1]?0:1;
    if(ans>min_ans[dir])query(node[pos].ch[dir],x);dir=!dir;
    if(ans>min_ans[dir])query(node[pos].ch[dir],x);
}
int query(Point x){
    ans=inf;
    query(root,x);
    return ans;
}
int main()
{
    int n=read_int(),m=read_int(),opt,x,y;
    Init(MAXN-1);
    _rep(i,1,n)
    temp[i].x=read_int(),temp[i].y=read_int();
    build(n);
    while(m--){
        opt=read_int(),x=read_int(),y=read_int();
        if(opt==1)
        Insert(Point(x,y));
        else
        enter(query(Point(x,y)));
    }
    return 0;
```

```
}
```

算法习题

K 远点对

洛谷p4357

题意

二维平面给定 n 个点，求第 k 远的点对。

题解

建树，然后对所有点查询，用小根堆维护前 $2k$ 大的数值即可。

```
const int MAXN=1e5+5,inf=0x7fffffff;
inline LL Pow(LL x){
    return x*x;
}
struct Point{
    LL x,y;
    Point(int x=0,int y=0):x(x),y(y){}
    LL get_dis(const Point &P){
        return Pow(P.x-x)+Pow(P.y-y);
    }
    void get_min(const Point &a,const Point &b){
        x=min(x,min(a.x,b.x));
        y=min(y,min(a.y,b.y));
    }
    void get_max(const Point &a,const Point &b){
        x=max(x,max(a.x,b.x));
        y=max(y,max(a.y,b.y));
    }
};
struct Node{
    int ch[2],cnt;
    Point p,r1,r2;
    void build(Point P){
        p=r1=r2=P;
        ch[0]=ch[1]=0;
        cnt=1;
    }
    LL get_value(Point P){
        return max(Pow(r1.x-P.x),Pow(r2.x-P.x))+max(Pow(r1.y-P.y),Pow(r2.y-
P.y));
    }
}node[MAXN];
void maintain(int pos){
```

```
node[pos].r1.get_min(node[node[pos].ch[0]].r1,node[node[pos].ch[1]].r1);
node[pos].r2.get_max(node[node[pos].ch[0]].r2,node[node[pos].ch[1]].r2);
    node[pos].cnt=node[node[pos].ch[0]].cnt+node[node[pos].ch[1]].cnt+1;
}
int pool[MAXN],pos1,pos2,root,dimension;
Point temp[MAXN];
bool cmp(const Point &p1,const Point &p2){
    if(!dimension)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);
}
void Init(int n){
    node[0].r1=Point(inf,inf);
    node[0].r2=Point(-inf,-inf);
    for(int i=n;i>=1;i--)
    pool[++pos1]=i;
}
void build(int &pos,int lef,int rig,bool d){
    if(lef>rig) return pos=0,void();
    pos=pool[pos1--];
    int mid=lef+rig>>1;
    dimension=d;
    nth_element(temp+lef,temp+mid,temp+rig+1,cmp);
    node[pos].p=node[pos].r1=node[pos].r2=temp[mid];
    build(node[pos].ch[0],lef,mid-1,!d);
    build(node[pos].ch[1],mid+1,rig,!d);
    maintain(pos);
}
void build(int n){build(root,1,n,false);}
priority_queue<LL,vector<LL>,greater<LL> >ans;
void update(LL v){
    if(ans.top()<v){
        ans.pop();
        ans.push(v);
    }
}
void query(int pos,Point x){
    if(!pos)
    return;
    LL max_ans[2];
    update(node[pos].p.get_dis(x));
    max_ans[0]=node[pos].ch[0]?node[node[pos].ch[0]].get_value(x):0;
    max_ans[1]=node[pos].ch[1]?node[node[pos].ch[1]].get_value(x):0;
    int dir=max_ans[0]>max_ans[1]?0:1;
    if(ans.top()<max_ans[dir])query(node[pos].ch[dir],x);dir=!dir;
    if(ans.top()<max_ans[dir])query(node[pos].ch[dir],x);
}
void query(Point x){query(root,x);}
int main()
{
    int n=read_int(),k=read_int(),x,y;
```

```
    _for(i,0,k<<1)
    ans.push(0LL);
    Init(MAXN-1);
    _rep(i,1,n)
    temp[i].x=read_int(),temp[i].y=read_int();
    build(n);
    _rep(i,1,n)
    query(temp[i]);
    enter(ans.top());
    return 0;
}
```

矩阵维护

洛谷p6514

题意

给定 n 个空串，编号为 $1 \sim n$接下来 q 个操作。

操作一：在编号为 $L \sim R$ 的字符串末尾插入一个字符。

操作二：查询编号为 $L \sim R$ 的字符串最长公共子序列长度。

题解

事实上对每个操作，将 L 视为第一维 R 视为第二维。

插入操作等价于插入点 (L,R)查询操作等价于查询矩阵$\left(1\sim L,R\sim n\right)$中点的个数。

对于查询操作，可以利用最小覆盖矩阵，类比线段树查询区间和操作。

```
const int MAXN=1e5+5,inf=1e9;
const double alpha=0.75;
struct Point{
    int x,y;
    Point(int x=0,int y=0):x(x),y(y){}
    bool in(int x1,int y1,int x2,int y2){return
x1<=x&&x<=x2&&y1<=y&&y<=y2;}
    void get_min(const Point &a,const Point &b){
        x=min(x,min(a.x,b.x));
        y=min(y,min(a.y,b.y));
    }
    void get_max(const Point &a,const Point &b){
        x=max(x,max(a.x,b.x));
        y=max(y,max(a.y,b.y));
    }
};
struct Node{
    int ch[2],cnt;
```

```
    Point p,r1,r2;
    void build(Point P){
        p=r1=r2=P;
        ch[0]=ch[1]=0;
        cnt=1;
    }
    bool in(int x1,int y1,int x2,int y2){return
x1<=r1.x&&y1<=r1.y&&x2>=r2.x&&y2>=r2.y;}
    bool out(int x1,int y1,int x2,int y2){return
x2<r1.x||y2<r1.y||x1>r2.x||y1>r2.y;}
}node[MAXN];
bool isbad(int pos){return
alpha*node[pos].cnt<max(node[node[pos].ch[0]].cnt,node[node[pos].ch[1]].cnt
)?true:false;}
void maintain(int pos){
node[pos].r1.get_min(node[node[pos].ch[0]].r1,node[node[pos].ch[1]].r1);
node[pos].r2.get_max(node[node[pos].ch[0]].r2,node[node[pos].ch[1]].r2);
    node[pos].cnt=node[node[pos].ch[0]].cnt+node[node[pos].ch[1]].cnt+1;
}
int pool[MAXN],pos1,pos2,root,dimension;
Point temp[MAXN];
bool cmp(const Point &p1,const Point &p2){
    if(!dimension)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);
}
void Init(int n){
    node[0].r1=Point(inf,inf);
    node[0].r2=Point(-inf,-inf);
    for(int i=n;i>=1;i--)
    pool[++pos1]=i;
}
void build(int &pos,int lef,int rig,bool d){
    if(lef>rig) return pos=0,void();
    pos=pool[pos1--];
    int mid=lef+rig>>1;
    dimension=d;
    nth_element(temp+lef,temp+mid,temp+rig+1,cmp);
    node[pos].p=node[pos].r1=node[pos].r2=temp[mid];
    build(node[pos].ch[0],lef,mid-1,!d);
    build(node[pos].ch[1],mid+1,rig,!d);
    maintain(pos);
}
void build(int n){build(root,1,n,false);}
void dfs(int pos){
    if(!pos)return;
    dfs(node[pos].ch[0]);
    pool[++pos1]=pos,temp[++pos2]=node[pos].p;
    dfs(node[pos].ch[1]);
}
void rebuild(int &pos,bool d){
```

```
    pos2=0;
    dfs(pos);
    build(pos,1,pos2,d);
}
void check(int &pos,Point x,bool d){
    if(pos){
        if(isbad(pos)){
            rebuild(pos,d);
            return;
        }
        dimension=d;
        if(cmp(node[pos].p,x))
        check(node[pos].ch[1],x,!d);
        else
        check(node[pos].ch[0],x,!d);
    }
}
void Insert(int &pos,Point x,bool d){
    if(!pos){
        pos=pool[pos1--];
        node[pos].build(x);
        return;
    }
    node[pos].cnt++;
    dimension=d;
    if(cmp(node[pos].p,x))Insert(node[pos].ch[1],x,!d);
    else Insert(node[pos].ch[0],x,!d);
    maintain(pos);
}
void Insert(Point x){
    Insert(root,x,false);
    check(root,x,false);
}
int query(int pos,int x1,int y1,int x2,int y2){
    if(!pos)
    return 0;
    if(node[pos].in(x1,y1,x2,y2))
    return node[pos].cnt;
    else if(node[pos].out(x1,y1,x2,y2))
    return 0;
    return
query(node[pos].ch[0],x1,y1,x2,y2)+query(node[pos].ch[1],x1,y1,x2,y2)+node[
pos].p.in(x1,y1,x2,y2);
}
int query(int x1,int y1,int x2,int y2){
    return query(root,x1,y1,x2,y2);
}
int main()
{
    int n=read_int(),m=read_int(),opt,x,y;
    Init(MAXN-1);
```

```
    while(m--){
        opt=read_int(),x=read_int(),y=read_int();
        if(opt==1)
        Insert(Point(x,y));
        else
        enter(query(1,y,x,n));
    }
    return 0;
}
```

优化技巧

洛谷p3810

题意

三维空间中给定 n 个点，编号为 $1 \sim n$，定义 $f[i]$ 表示恰好有 i 个元素满足 $x_i\lt x_j,y_i\lt y_j,z_i\lt z_j$ 且 $i\ne j$ 的 j 的个数。

要求输出 $f[0 \sim n-1]$。

题解

3D_Tree 时间复杂度过高，不可取，考虑降维。

将 z 作为第一关键字，x 作为第二关键字，y 作为第三关键字排序，可以把问题转换为 xOy 平面问题，做法和上题类似。

但是很遗憾，按上题的代码会 TLE，尝试做了些小优化，还是不太行，也可能是我代码不够优美。

放弃替罪羊树，考虑直接建树。将 (x,y) 点列去重后建树，每个结点 cnt 表示该位置的结点个数，初始值为 0。

插入查找维护每个结点 cnt 和 子树的 cnt 和，勉强 AC。

可以看出 KD_Tree 常数还是挺大的，要**慎用**。

```
const int MAXN=1e5+5,inf=1e9;
struct Pt{
    int a,b,c;
    bool operator < (const Pt &y)const{
        return c<y.c||(c==y.c&&a<y.a)||(c==y.c&&a==y.a&&b<y.b);
    }
    bool operator == (const Pt &y)const{
        return a==y.a&&b==y.b&&c==y.c;
    }
}A[MAXN],B[MAXN];
int dimension,q_x1,q_y1,q_x2,q_y2;
struct Point{
    int x,y;
```

```
    Point(int x=0,int y=0):x(x),y(y){}
    bool in(){return q_x1<=x&&x<=q_x2&&q_y1<=y&&y<=q_y2;}
    bool operator == (const Point &b)const{
        return x==b.x&&y==b.y;
    }
    int cmp(const Point &b){
        if(!dimension){
            if(x<b.x)return 0;
            else if(x==b.x){
                if(y<b.y)return 0;
                else if(y==b.y)return -1;
                else return 1;
            }
            else return 1;
        }
        else{
            if(y<b.y)return 0;
            else if(y==b.y){
                if(x<b.x)return 0;
                else if(x==b.x)return -1;
                else return 1;
            }
            else return 1;
        }
    }
    void get_min(const Point &a,const Point &b){
        x=min(x,min(a.x,b.x));
        y=min(y,min(a.y,b.y));
    }
    void get_max(const Point &a,const Point &b){
        x=max(x,max(a.x,b.x));
        y=max(y,max(a.y,b.y));
    }
};
struct Node{
    int ch[2],cnt,sz;
    Point p,r1,r2;
    bool in(){return q_x1<=r1.x&&q_y1<=r1.y&&q_x2>=r2.x&&q_y2>=r2.y;}
    bool out(){return q_x2<r1.x||q_y2<r1.y||q_x1>r2.x||q_y1>r2.y;}
}node[MAXN];
void maintain(int pos){
node[pos].r1.get_min(node[node[pos].ch[0]].r1,node[node[pos].ch[1]].r1);
node[pos].r2.get_max(node[node[pos].ch[0]].r2,node[node[pos].ch[1]].r2);
}
int pool[MAXN],pos1,pos2,root;
Point temp[MAXN],q_x;
bool cmp(const Point &p1,const Point &p2){
    if(!dimension)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);
}
void Init(int n){
```

```
    node[0].r1=Point(inf,inf);
    node[0].r2=Point(-inf,-inf);
    for(int i=n;i>=1;i--)
    pool[++pos1]=i;
}
void build(int &pos,int lef,int rig,bool d){
    if(lef>rig) return pos=0,void();
    pos=pool[pos1--];
    int mid=lef+rig>>1;
    dimension=d;
    nth_element(temp+lef,temp+mid,temp+rig+1,cmp);
    node[pos].p=node[pos].r1=node[pos].r2=temp[mid];
    build(node[pos].ch[0],lef,mid-1,!d);
    build(node[pos].ch[1],mid+1,rig,!d);
    maintain(pos);
}
void build(int n){build(root,1,n,false);}
void Insert(int pos,bool d){
    dimension=d;
    int dir=q_x.cmp(node[pos].p);
    if(dir==-1)
    node[pos].cnt++;
    else
    Insert(node[pos].ch[dir],!d);
    node[pos].sz++;
}
void Insert(Point x){
    q_x=x;
    Insert(root,false);
}
int query(int pos){
    if(!pos)
    return 0;
    if(node[pos].in())
    return node[pos].sz;
    else if(node[pos].out())
    return 0;
    return
query(node[pos].ch[0])+query(node[pos].ch[1])+node[pos].cnt*node[pos].p.in(
);
}
int query(int x1,int y1,int x2,int y2){
    q_x1=x1,q_y1=y1,q_x2=x2,q_y2=y2;
    return query(root);
}
int f[MAXN];
int main()
{
    int n=read_int(),k=read_int(),m,rep=0;
    _for(i,0,n)
```

```
    A[i].a=read_int(),A[i].b=read_int(),A[i].c=read_int();
    sort(A,A+n);
    _for(i,0,n)
    B[i]=A[i];
    m=unique(B,B+n)-B;
    _for(i,0,m){
        temp[i+1].x=B[i].a;
        temp[i+1].y=B[i].b;
    }
    sort(temp+1,temp+m+1,cmp);
    m=unique(temp+1,temp+m+1)-temp-1;
    Init(MAXN-1);
    build(m);

    for(int i=0,j=0;i<n;i++){
        if(A[i+1]==B[j]){
            Insert(Point(A[i].a,A[i].b));
            rep++;
            continue;
        }
        else{
            f[query(1,1,A[i].a,A[i].b)]+=rep+1;
            Insert(Point(A[i].a,A[i].b));
            j++;rep=0;
        }
    }
    _for(i,0,n)
    enter(f[i]);
    return 0;
}
```

From:
https://wiki.cvbbacm.com/ - **CVBB ACM Team**

Permanent link:
https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:kd_tree&rev=1595738285

Last update: **2020/07/26 12:38**