K-D Tree 在信息学竞赛中的应用

$n + e$

Tsinghua University

2016 年 7 月 31 日

简介

Q: K-D Tree 是啥我怎么没听说过？

A: K-D Tree 不仅是一个大模板，还是通用的骗分神器呢

Q: 我要学找哪里学？

A: 看下面的讲解吧……

1 基本用法
节点
构树
估价
插入
查询

2 例题

3 数据构造

1. 基本用法
    - 节点
    - 构树
    - 估价
    - 插入
    - 查询
2. 例题
3. 数据构造

- K-D Tree 中的一个结点，存储了一个 K 维空间域和一个 K 维的点坐标。
- 它的节点存储方式与 Splay Tree 有异曲同工之妙
- 你可以把 K 维空间域当作子树信息维护：K 维空间域就是子树内的所有节点的坐标范围

```
struct KDTree_Node{
    int d[MaxK],s[2],x[2],y[2],z[2],...;
}t[MaxN];
```

- d 存储一个点的所有维度上的坐标值
- s 存储儿子。s[0] 表示左儿子，s[1] 表示右儿子
- x 存储 x 坐标的范围。y, z 同理。当然可以弄成 a[MaxK][2]

```
struct KDTree_Node{
    int d[MaxK],s[2],x[2],y[2],z[2],...;
}t[MaxN];
```

- d 存储一个点的所有维度上的坐标值
- s 存储儿子。s[0] 表示左儿子，s[1] 表示右儿子
- x 存储 x 坐标的范围。y, z 同理。当然可以弄成 a[MaxK][2]

以二维平面为例，这样本来应该要叫 2-D Tree 的

```
struct KDTree_Node{
    int d[2],s[2],x[2],y[2],...;
}t[MaxN];
```

每个节点存储了一个二维平面上的点和一个二维平面（矩形区域）

1 基本用法
节点
构树
估价
插入
查询

2 例题

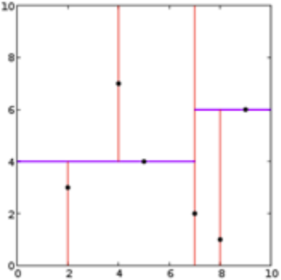
3 数据构造

algorithm 中的某个 stl

- 这里有 std::nth_element 的介绍。
- 简单来说，就是一个能在 $O(n)$ 时间内查询并定位长度为 n 的数组中第 k 小的元素，并将比它小的元素排在该元素之前，比它大的元素排在该元素之后。
- 具体实现类似快排

- K-D Tree 是一颗平衡二叉树，KD 即为 K-Dimension，每个节点即为一个 K 维的点。每个非叶节点可以想象为一个分割超平面，用垂直于坐标轴的超平面将空间分为两个部分，这样递归的从根节点不停的划分，直到没有点为止。
- 经典的构造 K-D Tree 的规则如下：
  1. 随着树的深度增加，循环的选取坐标轴，作为分割超平面的法向量。对于 3-D Tree 来说，根节点选取 x 轴，根节点的孩子选取 y 轴，根节点的孙子选取 z 轴，根节点的曾孙子选取 x 轴，这样循环下去。
  2. 每次均为当前点集中，选中某一维坐标的中位数的点作为切分点，切分点作为父节点，左右两侧为划分的作为左右两子树。
- 对于 n 个点的 K 维数据来说，建立 K-D Tree 的时间复杂度为 $O(Kn\log n)$。

- 例如存在 6 个点，分别为 $(2,3),(5,4),(9,6),(4,7),(8,1),(7,2)$，下图（引自 wiki）为平面对应的切分和对应的 K-D Tree。



```
int D;
struct P{int d[2];
    bool operator<(const P&a)const{return d[D]<a.d[D];}
}a[N];
struct T{int d[2],s[2],x[2],y[2];}t[N];
#define cmax(a,b) (a<b?a=b:a)
#define cmin(a,b) (a>b?a=b:a)
#define ls t[o].s[0]
#define rs t[o].s[1]
void mt(int f,int x){//maintain
    cmin(t[f].x[0],t[x].x[0]),cmax(t[f].x[1],t[x].x[1]);
    cmin(t[f].y[0],t[x].y[0]),cmax(t[f].y[1],t[x].y[1]);
}
```

```
int bt(int l,int r,int d){//build_tree
    D=d;int o=l+r>>1;
    std::nth_element(a+l,a+o,a+r+1);
    t[o].d[0]=t[o].x[0]=t[o].x[1]=a[o].d[0];
    t[o].d[1]=t[o].y[0]=t[o].y[1]=a[o].d[1];
    if(l<o)ls=bt(l,o-1,d^1),mt(o,ls);
    if(o<r)rs=bt(o+1,r,d^1),mt(o,rs);
    return o;
}
int main{}{
    ...
    root=bt(1,n,0);
}
```

- 调用 std :: nth_element 求中位数，是为了让两颗子树尽可能的平衡

- 调用 std :: nth_element 求中位数，是为了让两颗子树尽可能的平衡
- 其实，把 build_tree 函数第一行改成这样也是可以的

```
D=rand()&1;
```

- 为什么一定要按循环的顺序呢？嘿嘿嘿

1. 基本用法
   - 节点
   - 构树
   - 估价
   - 插入
   - 查询
2. 例题
3. 数据构造

- 这里的估价是算出目标点到当前查询区域距离的下界或上界
- 距离最小：一个点离当前域的最短距离（内部为 0）
- 距离最大：一个点离当前域的最长距离。显然会在端点取到

以平面为例，(x, y) 为查询的坐标，p 节点表示需估价的的矩形区域

- 曼哈顿最小

以平面为例，(x, y) 为查询的坐标，p 节点表示需估价的的矩形区域

- 曼哈顿最小

```
max(t[p].x[0]-x,0)+max(x-t[p].x[1],0)
+max(t[p].y[0]-y,0)+max(y-t[p].y[1],0)
```

- 曼哈顿最大

以平面为例，(x, y) 为查询的坐标，p 节点表示需估价的的矩形区域

- 曼哈顿最小

```
max(t[p].x[0]-x,0)+max(x-t[p].x[1],0)
+max(t[p].y[0]-y,0)+max(y-t[p].y[1],0)
```

- 曼哈顿最大

```
max(abs(x-t[p].x[1]),abs(t[p].x[0]-x))
+max(abs(y-t[p].y[1]),abs(t[p].y[0]-y))
```

- 欧几里德最小

- 欧几里德最小

```
sqr(max(max(x-t[p].x[1],t[p].x[0]-x),0))
+sqr(max(max(y-t[p].y[1],t[p].y[0]-y),0))
```

- 欧几里德最大

- 欧几里德最小

```
sqr(max(max(x-t[p].x[1],t[p].x[0]-x),0))
+sqr(max(max(y-t[p].y[1],t[p].y[0]-y),0))
```

- 欧几里德最大

```
max(sqr(x-t[p].x[0]),sqr(x-t[p].x[1]))
+max(sqr(y-t[p].y[0]),sqr(y-t[p].y[1]))
```

- 切比雪夫最小/最大

- 欧几里德最小

```
sqr(max(max(x-t[p].x[1],t[p].x[0]-x),0))
+sqr(max(max(y-t[p].y[1],t[p].y[0]-y),0))
```

- 欧几里德最大

```
max(sqr(x-t[p].x[0]),sqr(x-t[p].x[1]))
+max(sqr(y-t[p].y[0]),sqr(y-t[p].y[1]))
```

- 切比雪夫最小/最大？把坐标转 45° 之后就是曼哈顿距离了

1 基本用法
节点
构树
估价
插入
查询

2 例题

3 数据构造

- Splay 怎么插的，K-D Tree 也怎么插。

```
int main(){
    if(command=="Insert"){//(x,y)
        n++;scanf("%d%d",&x,&y);
        t[n].d[0]=t[n].x[0]=t[n].x[1]=x;
        t[n].d[1]=t[n].y[0]=t[n].y[1]=y;
        for(int p=root,D=0;p;D^=1){
            mt(p,n);
            int&nxt=t[p].s[t[n].d[D]>=t[p].d[D]];
            if(nxt==0){nxt=n;return;}else p=nxt;
        }
    }
}
```

一个 Trick

- 也可以在一开始把所有操作读进来，进行预处理
- 给 K-D Tree 上的每个节点打一个标记，表示该点是否被激活
- 插入时只需找到该点并修改标记即可

1 基本用法
- 节点
- 构树
- 估价
- 插入
- 查询

2 例题

3 数据构造

- 以查询与 (x, y) 最近的点（曼哈顿距离）与其的距离为例 BZOJ-2648/2716

```
#define oo 2147483647
#define getdist(p) max(t[p].x[0]-x,0)+max(x-t[p].x[1],0)\\
+max(t[p].y[0]-y,0)+max(y-t[p].y[1],0)
void query(int o){
    int tmp=abs(t[o].d[0]-x)+abs(t[o].d[1]-y),d[2];
    if(ls)d[0]=getdist(ls);else d[0]=oo;
    if(rs)d[1]=getdist(rs);else d[1]=oo;
    cmin(ans,tmp);tmp=d[0]>=d[1];
    if(d[tmp]<ans)query(t[o].s[tmp]);tmp^=1
    if(d[tmp]<ans)query(t[o].s[tmp]);
}
```

- 考虑暴力：就是把子树内的所有点 for 一遍过去
- getdist 函数就是之前所述的“估价函数”，计算点 (x, y) 与当前点范围的最小可能的差距有多少，哪个子树距离比较近就先走哪个。
- 如果当前区域内，连最小可能的距离都比当前的答案大，那么就认为，这块区域不可能再贡献答案了。因此这么做比暴力做法省去了许多访问节点的时间。
- 这种算法在随机数据上，单次操作复杂度是 $O(\log n)$ 的，但是在构造数据上约是 $O(\sqrt{n})$ 的。

- 考虑暴力：就是把子树内的所有点 for 一遍过去
- getdist 函数就是之前所述的“估价函数”，计算点 (x, y) 与当前点范围的最小可能的差距有多少，哪个子树距离比较近就先走哪个。
- 如果当前区域内，连最小可能的距离都比当前的答案大，那么就认为，这块区域不可能再贡献答案了。因此这么做比暴力做法省去了许多访问节点的时间。
- 这种算法在随机数据上，单次操作复杂度是 $O(\log n)$ 的，但是在构造数据上约是 $O(\sqrt{n})$ 的。
- How to 构造？

1 基本用法

2 例题
- K-D Tree 的重构
- K-D Tree 替代树套树
- K-D Tree 替代可持久化树套树
- K-D Tree 替代重量平衡树套线段树维护凸包
- 练习：K-D Tree 优化 Dp

3 数据构造

1 基本用法

2 例题
- K-D Tree 的重构
- K-D Tree 替代树套树
- K-D Tree 替代可持久化树套树
- K-D Tree 替代重量平衡树套线段树维护凸包
- 练习：K-D Tree 优化 Dp

3 数据构造

你有一个 $N \times N$ 的棋盘，每个格子内有一个整数，初始时的时候全部为 0，现在需要维护两种操作：

1. 将格子 (x, y) 里的数字加上 A
2. 输出 x_1 y_1 x_2 y_2 这个矩形内的数字和

$N \leq 500000$，操作数不超过 200000，强制在线，内存限制 20M

Source : BZOJ-4066

插入: 新建一个节点，坐标为 (x, y)，权值为 A，插入到 K-D Tree 中。

查询: 将之前的目标点改成目标矩形区域就好了。
如果接下来要查的区域完全被包含于目标矩形区域，那就跟线段树区间查询一样直接返回区域和就好了；
否则，如果接下来要查的区域与目标区域有交集，就往下查，否则不查。

插入： 新建一个节点，坐标为 (x, y)，权值为 A，插入到 K-D Tree 中。

查询： 将之前的目标点改成目标矩形区域就好了。
如果接下来要查的区域完全被包含于目标矩形区域，那就跟线段树区间查询一样直接返回区域和就好了；
否则，如果接下来要查的区域与目标区域有交集，就往下查，否则不查。

- 构造数据：在一个区域内拼命加点，让 K-D Tree 不平衡，从而导致插入/查询效率出现退化的情况，怎么办呢？

- 我们可以把替罪羊树的重构方法直接用到 K-D Tree 上：
- 在插入节点的时候，顺便再维护一个子树的 size 值
- 如果在某颗子树中，有

$$\max(size[leftson], size[rightson]) > size[p] \cdot fac$$

那么暴力重构以 p 为根的子树，使之变为平衡二叉树 ($fac \approx 0.5$)

- 这样做的复杂度并不会改变。详见 clj 论文。
- 一般来说，fac 取在 $0.65 \sim 0.75$ 之间；取值到了 0.9，说明你 K-D Tree 写挺了
- 代码详见此处

1 基本用法

2 例题

- K-D Tree 的重构
- K-D Tree 替代树套树
- K-D Tree 替代可持久化树套树
- K-D Tree 替代重量平衡树套线段树维护凸包
- 练习：K-D Tree 优化 Dp

3 数据构造

给定一棵以 1 为根的有根树，初始所有节点颜色为 1，每次将距离节点 a 不超过 d 的 a 的子节点染成 c，或询问点 a 的颜色
$n, m, c \leq 10^5$

Source : BZOJ-4154

- 将这棵树的 dfs 序写出来，并且求出每个点的深度。
- 将第 i 号点对应到二维平面上的点 $(start[i], deep[i])$
- 则修改操作等价于将横坐标在 $[start[a], end[a]]$ 内，纵坐标在 $[deep[a], deep[a]+d]$ 范围内的矩形区域的点的颜色都修改为 c
- 用支持标记下传的 K-D Tree 维护即可，时间复杂度 $O(n\log n + q\sqrt{n})$。
- 有没有很像二维线段树？是的这就是正解，不过由于这道题的特殊操作，我们可以只写 K-D Tree 就好了
- 代码详见此处

1. 基本用法
2. 例题
   - K-D Tree 的重构
   - K-D Tree 替代树套树
   - K-D Tree 替代可持久化树套树
   - K-D Tree 替代重量平衡树套线段树维护凸包
   - 练习：K-D Tree 优化 Dp
3. 数据构造

给出一个长度为 N 的序列，给出 M 个询问：在 $[l, r]$ 之间找到一个在这个区间里只出现过一次的数，并且要求找的这个数尽可能大。如果找不到这样的数，则直接输出 0。
$N \leq 100000$，$M \leq 200000$，$1 \leq a_i \leq N$，强制在线

Source : BZOJ-3489

- 记录每个位置的数前一次出现的位置 pre[i] 和后一次出现的位置 nxt[i]，然后我们询问的就是
  1. $l \le i \le r$
  2. pre[i]$< l$
  3. nxt[i]$> r$

  满足三个条件下的 $\max(a[i])$
- 将每个点的信息看作三维空间上带权值的点 (i, pre[i], nxt[i])，然后建立 K-D Tree。
- 询问的话，等价于第一维在 $[l, r]$ 范围内，第二维在 $[0, l-1]$ 范围内，第三维在 $[r+1, +\infty]$ 范围内的一个三维空间内，查询在里面的点权最大值。于是这样就能转换成 K-D Tree 啦 ~
- K 维的 K-D Tree 单次询问的复杂度是 $O(n^{1-\frac{1}{k}})$ 的，因此这样做的总复杂度为 $O(n^{\frac{5}{3}})$ 的。我拒绝写可持久化树套树。

1. 基本用法
2. 例题
   - K-D Tree 的重构
   - K-D Tree 替代树套树
   - K-D Tree 替代可持久化树套树
   - K-D Tree 替代重量平衡树套线段树维护凸包
   - 练习：K-D Tree 优化 Dp
3. 数据构造

平面上有 N 个点 $P_{1..N}$ 顺次相连，得到 $N-1$ 条线段。你需要支持以下操作：

1. 在某一个历史版本 T 的基础上，新建一个历史版本 T'，将一个新的点 P 插入到 P_i 和 P_{i+1} 之间，然后按照顺序对所有的点重新标记下标
2. 对于一个历史版本 T，给出一条直线，询问这条直线会与多少条线段相交。

$1 \le N, M \le 10^5$，所有的坐标范围 $\in [-10^8, 10^8]$，且每组数据中所有询问的答案总和不超过 10^6，插入操作的次数不会超过 5×10^4。注意这些线段可能会互相相交。强制在线。

Source : BZOJ-4056

- 正解是可持久化平衡树套可持久化平衡树维护动态凸包，或者标题，反正就是没人写的那种
- 然后用矩形框来拟合凸包，这样做的话忽略掉了底层凸包的信息，但是代码非常可写，虽然一堆 x=y 的点就能卡……
- 于是这题就变成了可持久化 Treap 维护矩形信息，中间过程用 K-D Tree 实现即可。
- 代码详见此处

1 基本用法

2 例题

K-D Tree 的重构
K-D Tree 替代树套树
K-D Tree 替代可持久化树套树
K-D Tree 替代重量平衡树套线段树维护凸包
练习：K-D Tree 优化 Dp

3 数据构造

给定 $N, A_{1..N}, B_{1..N}$，分析题目以后会得到如下的 Dp 方程：

$$F_i = \min\left\{\sqrt{{F_j}^2 + (A_i - A_j)^2} \;\middle|\; (0 \le j < i,\ A_i \ge B_j)\right\}$$

$N \le 10^5,\ 1 \le A_i \le 10^6,\ 0 \le B_i \le 10^6$

Source : Xj Noi2015 训练 19 不可视境界线

- 如果不用 K-D Tree 的话，好像要 cdq 分治求凸包才能搞
- 考虑如何用 K-D Tree 解决。只需将第 i 号点对应到二维平面上的点 $(A_i, {F_i}^2)$，每次询问之后再插入。
- 定义距离为 $(x_1 - x_2)^2 + y_1 - y_2$，每个节点上的矩形区域 y 范围最小值均为 0
- 对应的估价函数应改为 $y[p] + \max\left(x - x_1[p], x_0[p] - x, 0\right)^2$
- 代码详见此处

1 基本用法

2 例题

3 数据构造
- 曼哈顿距离
- 欧几里德距离

- K-D Tree 的效率很大程度上取决于估价函数的优劣。
- 如果要想把 K-D Tree 卡 T 掉，就要从估价函数上入手。

1 基本用法

2 例题

3 数据构造
曼哈顿距离
欧几里德距离

- 还是以询问定点到点集的最小曼哈顿距离为例
- 只需将点集中的点放到 $y = x + C$ 上即可，这样的话所有点都能成为最优点。
- 由于估价函数的不足，K-D Tree 会将所有的子树遍历一遍，从而导致效率的退化。

1 基本用法

2 例题

3 数据构造
曼哈顿距离
欧几里德距离

- 只需将点集中的点放到 $(x-x_0)^2+(y-y_0)^2=r^2$ 上即可，这样的话所有点都有可能成为最优点。