原创

【luogu 1429】平面最近点对 K-D Tree

2019-09-19 11:26:14 我是一只计算鸡 阅读数 9 更多

KD树用来维护高维空间状态信息。非叶子结点都是记录在那个维度上的切割。

K-D Tree模板题，记录一下。

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cmath>
using namespace std;
const int maxn = 200000 + 7;
const double INF = 2e18;
double ans = INF;
int n;  // 结点个数
int d[maxn]; // 每个结点所在的维度
int lc[maxn]; // 结点的左子结点
int rc[maxn]; // 结点的右子结点
struct node {
        double x; // 横坐标
        double y;  // 纵坐标
}s[maxn]; // 结点
double L[maxn]; // 左区间
double R[maxn]; // 右区间
double U[maxn]; // 上区间
double D[maxn]; // 下区间
double dist(int a, int b)  // 求出两个点的欧几里得距离
{
        return (s[a].x - s[b].x) * (s[a].x - s[b].x) + (s[a].y - s[b].y) * (s[a].y - s[b].y);
}
bool cmp1 (node a, node b)  // 在 x 维度上排序
{
        return a.x < b.x;
}
bool cmp2 (node a, node b)  // 在 y 维度上排序
{
        return a.y < b.y;
}
void maintain(int x) // 维护结点信息
{
        L[x] = R[x] = s[x].x;  // 每个矩形的左右开始为自己的横坐标
        D[x] = U[x] = s[x].y;  // 每个矩形的上下开始为自己的纵坐标
        if(lc[x]) {  // 如果 x 有左儿子
                L[x] = min(L[x], L[lc[x]]); // 父矩形的左端点比左儿子矩形的左端点小
                R[x] = max(R[x], R[lc[x]]); // 父矩形的右端点比左儿子矩形的右端点大
                D[x] = min(D[x], D[lc[x]]); // 父矩形的下端点比左儿子矩形的下端点小
                U[x] = max(U[x], U[lc[x]]); // 父矩形的上端点比左儿子矩形的上端点大
        }
        if(rc[x]) { // 如果 x 有右儿子
                L[x] = min(L[x], L[rc[x]]); // 父矩形的左端点比右儿子矩形的左端点小
                R[x] = max(R[x], R[rc[x]]); // 父矩形的右端点比右儿子矩形的右端点大
                D[x] = min(D[x], D[rc[x]]); // 父矩形的下端点比右儿子矩形的下端点小
                U[x] = max(U[x], U[rc[x]]); // 父矩形的上端点比右儿子矩形的上端点大
        }
}
int build(int l, int r) // 建树
{
        if(l >= r) return 0; // 叶子结点 或者 不合法
        int mid = (l + r) >> 1;  // 分治
        double avx = 0;  // 区间 (l, r) 上 x 的平均值
        double avy = 0;  // 区间 (l, r) 上 y 的平均值
        double vax = 0;  // 区间 (l, r) 上 x 的方差
        double vay = 0;  // 区间 (l, r) 上 y 的方差
        for(int i = l; i <= r; i++) { // 分别对 x y 求和
```

```
59                  avx += s[i].x;
60                  avy += s[i].y;
61          }
62          avx /= (double)(r - l + 1);  // 计算 x 的平均值
63          avy /= (double)(r - l + 1);  // 计算 y 的平均值
64          for(int i = l; i <= r; i++) {  // 计算方差 = 偏差的平方的平均数
65                  vax += (s[i].x - avx) * (s[i].x - avx);  // 计算 x 的方差
66                  vay += (s[i].y - avy) * (s[i].y - avy);  // 计算 y 的方差
67          }
68          if(vax >= vay) { // 方差大 点稀疏
69                  d[mid] = 1; // x 方差大 选择第一维度
70                  nth_element(s + l, s + mid, s + r + 1, cmp1);  // 找(l, r)中 x 的中位数
71                  // 排序后 s[mid] 左边的值小于 s[mid]
72                  // s[mid] 右边的值大于 s[mid]
73          }
74          else {
75                  d[mid] = 2; // y 方差大 选择第二维度
76                  nth_element(s + l, s + mid, s + r + 1, cmp2); // 找(l, r)中 y 的中位数
77          }
78          lc[mid] = build(l, mid - 1);  // 递归建立左子树
79          rc[mid] = build(mid + 1, r);  // 递归建立右子树
80          maintain(mid);  // 向上更新
81          return mid;
82  }
83  double f(int a, int b)  // 返回 a 点到 b点 所在矩形的距离
84  {
85          double ret = 0;
86          if(L[b] > s[a].x) { // a 点在 b 矩形的左边
87                  ret += (L[b] - s[a].x) * (L[b] - s[a].x);
88          }
89          if(R[b] < s[a].x) { // a 点在 b 矩形的右边
90                  ret += (s[a].x - R[b]) * (s[a].x - R[b]);
91          }
92          if(D[b] > s[a].y) {  // a 点在 b 矩形的下面
93                  ret += (D[b] - s[a].y) * (D[b] - s[a].y);
94          }
95          if(U[b] < s[a].y) { // a 点在 b 矩形的上面
96                  ret += (s[a].y - U[b]) * (s[a].y - U[b]);
97          }
98          return ret;
99  }
100 void query(int l, int r, int x)  // (l, r)区间 与 x 点最近的距离
101 {
102         if(l > r) return ; // 不合法
103         int mid = (l + r) >> 1; // 分治
104         if(mid != x) ans = min(ans, dist(x, mid)); // 如果中点不是 x 更新 最小距离
105         if(l == r) return ; // 叶子结点 存储具体的点 上面已经更新了叶子结点到x的距离
106         double distl = f(x, lc[mid]);  // x点到左儿子矩形的距离
107         double distr = f(x, rc[mid]);  // x点到右儿子矩形的距离
108         if(distl < ans && distr < ans) {  // x 到左右儿子矩形的距离都比之前答案小
109                 if(distl < distr) { // 到左儿子矩形的距离小于到右儿子矩形的面积
110                         query(l, mid - 1, x);  // 递归搜索左子树
111                         if(distr < ans) {
112                                 query(mid + 1, r, x); // 如果 x 到右边矩形的距离比答案距离小 递归搜索右子树
113                         }
114                 }
115                 else { // 到右儿子矩形的距离小于到左儿子矩形的距离
116                         query(mid + 1, r, x); // 递归搜索右子树
117                         if(distl < ans) {
118                                 query(l, mid - 1, x); // 如果 x 到左边矩形的面积比答案距离小 递归搜索左子树
119                         }
120                 }
121         }
122         else {
123                 if(distl < ans) query(l, mid - 1, x);  // 只有 x 到左儿子矩形的距离小于答案 递归搜索左子树
124                 if(distr < ans) query(mid + 1, r, x); // 只有 x 到右儿子矩形的距离小于答案 递归所搜右子树
125         }
126 }
127 int main()
128 {
129         ans = INF;
```

```
130         scanf("%d", &n);
131         for(int i = 1; i <= n; i++) scanf("%lf %lf", &s[i].x, &s[i].y);
132         build(1, n);
133         for(int i = 1; i <= n; i++) query(1, n, i);
134         printf("%.4f\n", sqrt(ans));
135         return 0;
136 }
```


文章最后发布于: 2019-09-19 11:30:38