

# K-D Tree

k-D Tree(KDT, k-Dimension Tree) 是一种可以 **高效处理  $k$  维空间信息** 的数据结构。

在结点数  $n$  远大于  $2^k$  时, 应用 k-D Tree 的时间效率很好。

在算法竞赛的题目中, 一般有  $k = 2$ 。在本页面分析时间复杂度时, 将认为  $k$  是常数。

## 建树

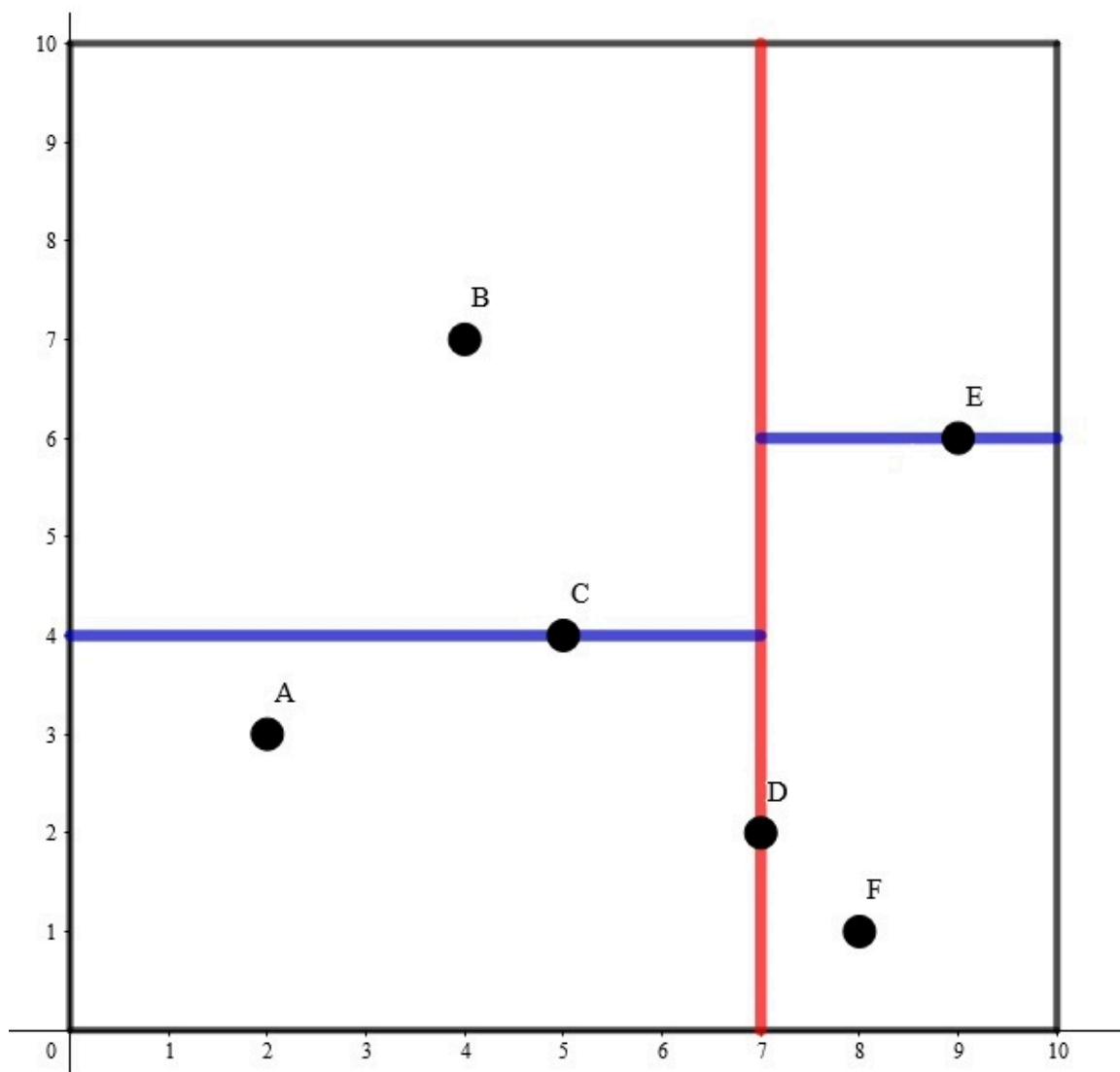
k-D Tree 具有二叉搜索树的形态, 二叉搜索树上的每个结点都对应  $k$  维空间内的一个点。其每个子树中的点都在一个  $k$  维的超长方体内, 这个超长方体内的所有点也都在这个子树中。

假设我们已经知道了  $k$  维空间内的  $n$  个不同的点的坐标, 要将其构建成一棵 k-D Tree, 步骤如下:

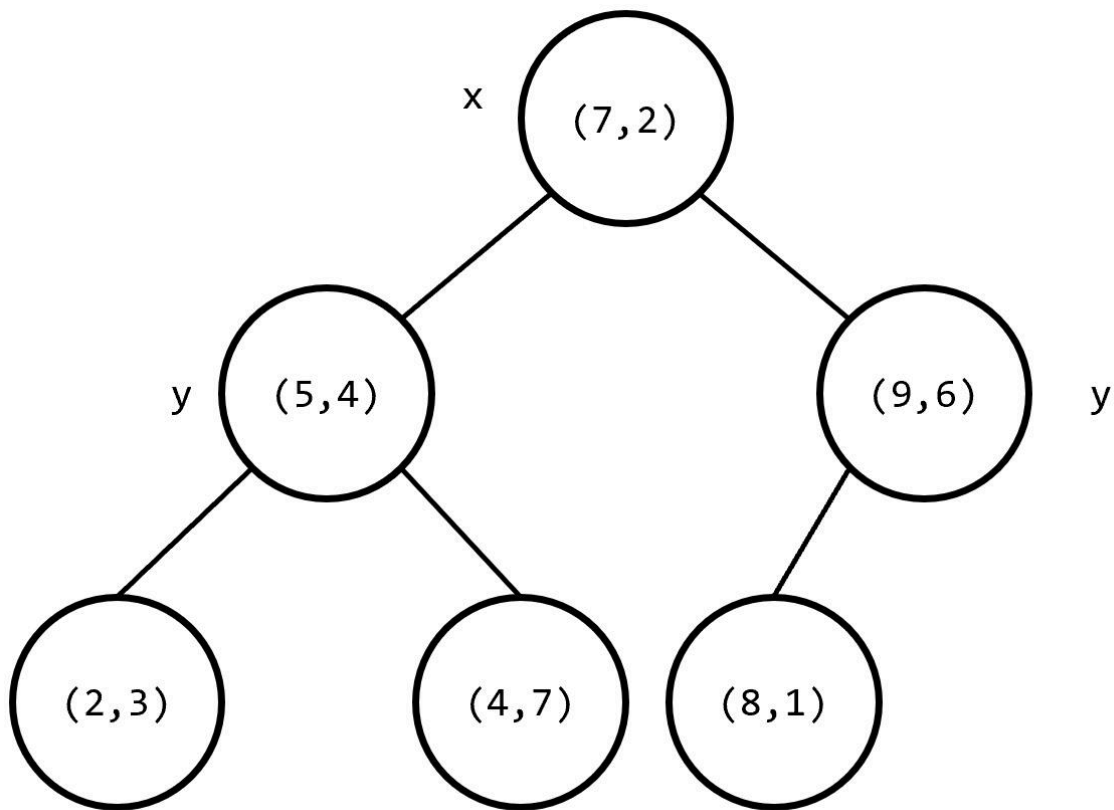
1. 若当前超长方体中只有一个点, 返回这个点。
2. 选择一个维度, 将当前超长方体按照这个维度分成两个超长方体。
3. 选择切割点: 在选择的维度上选择一个点, 这一维度上的值小于这个点的归入一个超长方体 (左子树), 其余的归入另一个超长方体 (右子树)。
4. 将选择的点作为这棵子树的根节点, 递归对分出的两个超长方体构建左右子树, 维护子树的信息。

为了方便理解, 我们举一个  $k = 2$  时的例子。





其构建出 k-D Tree 的形态可能是这样的：



其中树上每个结点上的坐标是选择的分割点的坐标，非叶子结点旁的  $x$  或  $y$  是选择的切割维度。

这样的复杂度无法保证。对于 2, 3 两步，我们提出两个优化：

1. 轮流选择  $k$  个维度，以保证在任意连续  $k$  层里每个维度都被切割到。
2. 每次在维度上选择切割点时选择该维度上的 **中位数**，这样可以保证每次分成的左右子树大小尽量相等。

可以发现，使用优化 2 后，构建出的 k-D Tree 的树高最多为  $\log n + O(1)$ 。

现在，构建 k-D Tree 时间复杂度的瓶颈在于快速选出一个维度上的中位数，并将在该维度上的值小于该中位数的置于中位数的左边，其余置于右边。如果每次都使用 `sort` 函数对该维度进行排序，时间复杂度是  $O(n \log^2 n)$  的。事实上，单次找出  $n$  个元素中的中位数并将中位数置于排序后正确的位置的复杂度可以达到  $O(n)$ 。

我们来回顾一下快速排序的思想。每次我们选出一个数，将小于该数的置于该数的左边，大于该数的置于该数的右边，保证该数在排好序后正确的位置上，然后递归排序左侧和右侧的值。这样的期望复杂度是  $O(n \log n)$  的。但是由于 k-D Tree 只要求要中位数在排序后正确的位置上，所以我们只需要递归排序包含中位数的一侧。可以证明，这样的期望复杂度是  $O(n)$  的。在 `algorithm` 库中，有一个实现相同功能的函数 `nth_element()`，要找到 `s[l]` 和 `s[r]` 之间的值按照排序规则 `cmp` 排序后在 `s[mid]` 位置上的值，并保证 `s[mid]` 左边的值小于 `s[mid]`，右边的值大于 `s[mid]`，只需写 `nth_element(s+l, s+mid, s+r+1, cmp)`。

借助这种思想，构建 k-D Tree 时间复杂度是  $O(n \log n)$  的。

## 高维空间上的操作

在查询高维矩形区域内的所有点的一些信息时，记录每个结点子树内每一维度上的坐标的最大值和最小值。如果当前子树对应的矩形与所求矩形没有交点，则不继续搜索其子树；如果当前子树对应的矩形完全包含在所求矩形内，返回当前子树内所有点的权值和；否则，判断当前点是否在所求矩形内，更新答案并递归在左右子树中查找答案。



实现



```
1  int query(int p) {
2      if (!p) return 0;
3      bool flag{false};
4      for (int k : {0, 1}) flag |= (!l.x[k] <= t[p].L[k] &&
5  t[p].R[k] <= h.x[k]);
6      if (!flag) return t[p].sum;
7      for (int k : {0, 1})
8          if (t[p].R[k] < l.x[k] || h.x[k] < t[p].L[k]) return 0;
9      int ans{0};
10     flag = false;
11     for (int k : {0, 1}) flag |= (!l.x[k] <= t[p].x[k] &&
12 t[p].x[k] <= h.x[k]);
13     if (!flag) ans = t[p].v;
14     return ans += query(t[p].l) + query(t[p].r);
15 }
```

## 复杂度分析

先考虑二维的，在查询矩形  $R$  时，我们将  $k$ -D Tree 上的结点分为三类：

1. 与  $R$  无交。
2. 完全被  $R$  包含。
3. 部分被  $R$  包含。

显然单次查询的复杂度是第 3 类点的个数。注意到第三类点的矩形要么完全包含  $R$ ，要么互不包含，而前者显然只有  $O(h) = O(\log n)$  个，现在我们来分析后者的个数。

首先，我们不妨令矩形的所有边偏移  $\epsilon$ ，使得查询矩形不穿过已经有的任何点。这样显然是不影响矩形的查询所涵盖的点集的。

注意到互不包含的第 3 类点所对应的矩形，一定有  $R$  的一条边穿过之。所以我们只需要计算  $R$  的每条边穿过的矩形个数，即任意一条线段最多经过多少个点对应的矩形。

考虑对于某一个结点  $u$ ，它有四个孙子，且它到每一个孙子都在两个维度上各进行了一次划分。经过观察可以发现，按照这种方法将一个矩形划分成四个子矩形，一条与坐标轴平行的线段最多经过两个区域，即从  $u$  出发的查询，最多向下进入两个孙子仍有第 3 类点（如果线段刚好与分割边界重合则不一定，但是我们偏移查询矩形边界的操作使得这种情况不存在）。

而因为建树的时候，每个点是其整个子树在当前划分维度上的中位数，所以子树大小必定减半。于是，设  $u$  的子树大小为  $n$ ，我们能写出如下递归式：

$$T(n) = 2T(n/4) + O(1)$$

由主定理得  $T(n) = O(\sqrt{n})$ 。

将递归式推广到  $k$  维，即  $T(n) = 2^{k-1}T(n/2^k) + O(1)$ ，于是  $T(n) = O(n^{1-\frac{1}{k}})$ （将  $k$  视为常数）。

## 插入/删除

如果维护的这个  $k$  维点集是可变的，即可能会插入或删除一些点，此时  $k$ -D Tree 的平衡性无法保证。由于  $k$ -D Tree 的构造，不能支持旋转，类似与 FHQ Treap 的随机优先级也不能保证其复杂度。对此，有两种比较常见的维护方法。



### Note



很多选手会使用替罪羊树结构来维护。但是注意到在刚才的复杂度分析中，要求儿子的子树大小严格减半，即树高必须为严格的  $\log n + O(1)$ ，而替罪羊树只满足树高  $O(\log n)$ ，故查询复杂度无法保证。

## 根号重构

插入的时候，先存下来要插入的点，每  $B$  次插入进行一次重构。

删除打个标记即可。如果要求较为严格，可以维护树内有多少个被删除了，达到  $B$  则重构。

修改复杂度均摊  $O(n \log n / B)$ ，查询  $O(B + n^{1-\frac{1}{k}})$ ，若二者数量同阶则  $B = O(\sqrt{n \log n})$  最优（修改  $O(\sqrt{n \log n})$ ，查询  $O(\sqrt{n \log n} + n^{1-\frac{1}{k}})$ ）。

## 二进制分组

考虑维护若干棵 2 的自然数次幂的  $k$ -D Tree，满足这些树的大小之和为  $n$ 。

插入的时候，新增一棵大小为 1 的  $k$ -D Tree，然后不断将相同大小的树合并（直接拍扁重构）。实现的时候可以只重构一次。

容易发现需要合并的树的大小一定从  $2^0$  开始且指数连续。复杂度类似二进制加法，是均摊  $O(n \log^2 n)$  的，因为重构本身带  $\log$ 。

查询的时候，直接分别在每颗树上查询，复杂度为  $O\left(\sum_{i \geq 0} \left(\frac{n}{2^i}\right)^{1-\frac{1}{k}}\right) = O(n^{1-\frac{1}{k}})$ 。

## 例题



### 洛谷 P4148 简单题



在一个初始值全为 0 的  $n \times n$  的二维矩阵上，进行  $q$  次操作，每次操作为以下两种之一：

1.  $1 \ x \ y \ A$ ：将坐标  $(x, y)$  上的数加上  $A$ 。
2.  $2 \ x1 \ y1 \ x2 \ y2$ ：输出以  $(x1, y1)$  为左下角， $(x2, y2)$  为右上角的矩形内（包括矩形边界）的数字和。

强制在线。内存限制 20M。保证答案及所有过程量在 `int` 范围内。

$$1 \leq n \leq 500000, 1 \leq q \leq 200000$$

20M 的空间卡掉了所有树套树，强制在线卡掉了 CDQ 分治，只能使用 k-D Tree。

以下是二进制分组的参考代码。



## 参考代码



```
1  #include <bits/stdc++.h>
2  using namespace std;
3  constexpr int N(2e5), LG{18};
4
5  struct pt {
6      int x[2];
7      int v, sum;
8      int l, r;
9      int L[2], R[2];
10 } t[N + 5], l, h;
11
12 int rt[LG];
13 int b[N + 5], cnt;
14
15 void upd(int p) {
16     t[p].sum = t[t[p].l].sum + t[t[p].r].sum + t[p].v;
17     for (int k : {0, 1}) {
18         t[p].L[k] = t[p].R[k] = t[p].x[k];
19         if (t[p].l) {
20             t[p].L[k] = min(t[p].L[k], t[t[p].l].L[k]);
21             t[p].R[k] = max(t[p].R[k], t[t[p].l].R[k]);
22         }
23         if (t[p].r) {
24             t[p].L[k] = min(t[p].L[k], t[t[p].r].L[k]);
25             t[p].R[k] = max(t[p].R[k], t[t[p].r].R[k]);
26         }
27     }
28 }
29
30 int build(int l, int r, int dep = 0) {
31     int p[l + r >> 1];
32     nth_element(b + l, b + p, b + r + 1,
33         [dep](int x, int y) { return t[x].x[dep] <
34 t[y].x[dep]; });
35     int x{b[p]};
36     if (l < p) t[x].l = build(l, p - 1, dep ^ 1);
37     if (p < r) t[x].r = build(p + 1, r, dep ^ 1);
38     upd(x);
39     return x;
40 }
41
42 void append(int &p) {
43     if (!p) return;
44     b[++cnt] = p;
45     append(t[p].l);
46     append(t[p].r);
47     p = 0;
48 }
49
```

```

50 int query(int p) {
51     if (!p) return 0;
52     bool flag{false};
53     for (int k : {0, 1}) flag |= (!(l.x[k] <= t[p].L[k] &&
54 t[p].R[k] <= h.x[k]));
55     if (!flag) return t[p].sum;
56     for (int k : {0, 1})
57         if (t[p].R[k] < l.x[k] || h.x[k] < t[p].L[k]) return 0;
58     int ans{0};
59     flag = false;
60     for (int k : {0, 1}) flag |= (!(l.x[k] <= t[p].x[k] &&
61 t[p].x[k] <= h.x[k]));
62     if (!flag) ans = t[p].v;
63     return ans += query(t[p].l) + query(t[p].r);
64 }
65
66 int main() {
67     int n;
68     cin >> n;
69     int lst{0};
70     n = 0;
71     while (true) {
72         int op;
73         cin >> op;
74         if (op == 1) {
75             int x, y, A;
76             cin >> x >> y >> A;
77             x ^= lst;
78             y ^= lst;
79             A ^= lst;
80             t[++n] = {{x, y}, A};
81             b[cnt = 1] = n;
82             for (int sz{0};; ++sz)
83                 if (!rt[sz]) {
84                     rt[sz] = build(1, cnt);
85                     break;
86                 } else
87                     append(rt[sz]);
88         } else if (op == 2) {
89             cin >> l.x[0] >> l.x[1] >> h.x[0] >> h.x[1];
90             l.x[0] ^= lst;
91             l.x[1] ^= lst;
92             h.x[0] ^= lst;
93             h.x[1] ^= lst;
94             lst = 0;
95             for (int i{0}; i < LG; ++i) lst += query(rt[i]);
96             cout << lst << "\n";
97         } else
98             break;
99     }

```



```
    return 0;
}
```

## 邻域查询



### Warning



使用 k-D Tree 单次查询最近点的时间复杂度最坏还是  $O(n)$  的，但不失为一种优秀的骗分算法，使用时请注意。在这里对邻域查询的讲解仅限于加强对 k-D Tree 结构的认识。



### 例题 luogu P1429 平面最近点对（加强版）



给定平面上的  $n$  个点  $(x_i, y_i)$ ，找出平面上最近两个点对之间的 [欧几里得距离](#)。

$2 \leq n \leq 200000, 0 \leq x_i, y_i \leq 10^9$

首先建出关于这  $n$  个点的 2-D Tree。

枚举每个结点，对于每个结点找到不等于该结点且距离最小的点，即可求出答案。每次暴力遍历 2-D Tree 上的每个结点的时间复杂度是  $O(n)$  的，需要剪枝。我们可以维护一个子树中的所有结点在每一维上的坐标的最小值和最大值。假设当前已经找到的最近点对的距离是  $ans$ ，如果查询点到子树内所有点都包含在内的长方形的 **最近** 距离大于等于  $ans$ ，则在这个子树内一定没有答案，搜索时不进入这个子树。

此外，还可以使用一种启发式搜索的方法，即若一个结点的两个子树都有可能包含答案，先在与查询点距离最近的一个子树中搜索答案。可以认为，**查询点到子树对应的长方形的最近距离就是此题的估价函数**。



## 参考代码



```
1  #include <algorithm>
2  #include <cmath>
3  #include <cstdio>
4  #include <cstdlib>
5  #include <cstring>
6  using namespace std;
7  const int maxn = 200010;
8  int n, d[maxn], lc[maxn], rc[maxn];
9  double ans = 2e18;
10
11 struct node {
12     double x, y;
13 } s[maxn];
14
15 double L[maxn], R[maxn], D[maxn], U[maxn];
16
17 double dist(int a, int b) {
18     return (s[a].x - s[b].x) * (s[a].x - s[b].x) +
19           (s[a].y - s[b].y) * (s[a].y - s[b].y);
20 }
21
22 bool cmp1(node a, node b) { return a.x < b.x; }
23
24 bool cmp2(node a, node b) { return a.y < b.y; }
25
26 void maintain(int x) {
27     L[x] = R[x] = s[x].x;
28     D[x] = U[x] = s[x].y;
29     if (lc[x])
30         L[x] = min(L[x], L[lc[x]]), R[x] = max(R[x], R[lc[x]]),
31         D[x] = min(D[x], D[lc[x]]), U[x] = max(U[x], U[lc[x]]);
32     if (rc[x])
33         L[x] = min(L[x], L[rc[x]]), R[x] = max(R[x], R[rc[x]]),
34         D[x] = min(D[x], D[rc[x]]), U[x] = max(U[x], U[rc[x]]);
35 }
36
37 int build(int l, int r) {
38     if (l > r) return 0;
39     if (l == r) {
40         maintain(l);
41         return l;
42     }
43     int mid = (l + r) >> 1;
44     double avx = 0, avy = 0, vax = 0, vay = 0; // average variance
45     for (int i = l; i <= r; i++) avx += s[i].x, avy += s[i].y;
46     avx /= (double)(r - l + 1);
47     avy /= (double)(r - l + 1);
48     for (int i = l; i <= r; i++)
49         vax += (s[i].x - avx) * (s[i].x - avx),
```

```

50         vay += (s[i].y - avy) * (s[i].y - avy);
51     if (vax >= vay)
52         d[mid] = 1, nth_element(s + l, s + mid, s + r + 1, cmp1);
53     else
54         d[mid] = 2, nth_element(s + l, s + mid, s + r + 1, cmp2);
55     lc[mid] = build(l, mid - 1), rc[mid] = build(mid + 1, r);
56     maintain(mid);
57     return mid;
58 }
59
60 double f(int a, int b) {
61     double ret = 0;
62     if (L[b] > s[a].x) ret += (L[b] - s[a].x) * (L[b] - s[a].x);
63     if (R[b] < s[a].x) ret += (s[a].x - R[b]) * (s[a].x - R[b]);
64     if (D[b] > s[a].y) ret += (D[b] - s[a].y) * (D[b] - s[a].y);
65     if (U[b] < s[a].y) ret += (s[a].y - U[b]) * (s[a].y - U[b]);
66     return ret;
67 }
68
69 void query(int l, int r, int x) {
70     if (l > r) return;
71     int mid = (l + r) >> 1;
72     if (mid != x) ans = min(ans, dist(x, mid));
73     if (l == r) return;
74     double distl = f(x, lc[mid]), distr = f(x, rc[mid]);
75     if (distl < ans && distr < ans) {
76         if (distl < distr) {
77             query(l, mid - 1, x);
78             if (distr < ans) query(mid + 1, r, x);
79         } else {
80             query(mid + 1, r, x);
81             if (distl < ans) query(l, mid - 1, x);
82         }
83     } else {
84         if (distl < ans) query(l, mid - 1, x);
85         if (distr < ans) query(mid + 1, r, x);
86     }
87 }
88
89 int main() {
90     scanf("%d", &n);
91     for (int i = 1; i <= n; i++) scanf("%lf%lf", &s[i].x, &s[i].y);
92     build(1, n);
93     for (int i = 1; i <= n; i++) query(1, n, i);
94     printf("%.4lf\n", sqrt(ans));
95     return 0;
96 }

```



### 例题 「CQOI2016」K 远点对



给定平面上的  $n$  个点  $(x_i, y_i)$ ，求欧几里得距离下的第  $k$  远无序点对之间的距离。

$$n \leq 100000, 1 \leq k \leq 100, 0 \leq x_i, y_i < 2^{31}$$

和上一道例题类似，从最近点对变成了  $k$  远点对，估价函数改成了查询点到子树对应的长方形区域的最远距离。用一个小根堆来维护当前找到的前  $k$  远点对之间的距离，如果当前找到的点对距离大于堆顶，则弹出堆顶并插入这个距离，同样的，使用堆顶的距离来剪枝。

由于题目中强调的是无序点对，即交换前后两点的顺序后仍是相同的点对，则每个有序点对会被计算两次，那么读入的  $k$  要乘以 2。



## 参考代码



```
1  #include <algorithm>
2  #include <cstring>
3  #include <iostream>
4  #include <queue>
5  using namespace std;
6  const int maxn = 100010;
7  long long n, k;
8  priority_queue<long long, vector<long long>, greater<long long> >
9  q;
10
11 struct node {
12     long long x, y;
13 } s[maxn];
14
15 bool cmp1(node a, node b) { return a.x < b.x; }
16
17 bool cmp2(node a, node b) { return a.y < b.y; }
18
19 long long lc[maxn], rc[maxn], L[maxn], R[maxn], D[maxn], U[maxn];
20
21 void maintain(int x) {
22     L[x] = R[x] = s[x].x;
23     D[x] = U[x] = s[x].y;
24     if (lc[x])
25         L[x] = min(L[x], L[lc[x]]), R[x] = max(R[x], R[lc[x]]),
26         D[x] = min(D[x], D[lc[x]]), U[x] = max(U[x], U[lc[x]]);
27     if (rc[x])
28         L[x] = min(L[x], L[rc[x]]), R[x] = max(R[x], R[rc[x]]),
29         D[x] = min(D[x], D[rc[x]]), U[x] = max(U[x], U[rc[x]]);
30 }
31
32 int build(int l, int r) {
33     if (l > r) return 0;
34     int mid = (l + r) >> 1;
35     double av1 = 0, av2 = 0, va1 = 0, va2 = 0; // average variance
36     for (int i = l; i <= r; i++) av1 += s[i].x, av2 += s[i].y;
37     av1 /= (r - l + 1);
38     av2 /= (r - l + 1);
39     for (int i = l; i <= r; i++)
40         va1 += (av1 - s[i].x) * (av1 - s[i].x),
41         va2 += (av2 - s[i].y) * (av2 - s[i].y);
42     if (va1 > va2)
43         nth_element(s + l, s + mid, s + r + 1, cmp1);
44     else
45         nth_element(s + l, s + mid, s + r + 1, cmp2);
46     lc[mid] = build(l, mid - 1);
47     rc[mid] = build(mid + 1, r);
48     maintain(mid);
49     return mid;
```

```

50 }
51
52 long long sq(long long x) { return x * x; }
53
54 long long dist(int a, int b) {
55     return max(sq(s[a].x - L[b]), sq(s[a].x - R[b])) +
56            max(sq(s[a].y - D[b]), sq(s[a].y - U[b]));
57 }
58
59 void query(int l, int r, int x) {
60     if (l > r) return;
61     int mid = (l + r) >> 1;
62     long long t = sq(s[mid].x - s[x].x) + sq(s[mid].y - s[x].y);
63     if (t > q.top()) q.pop(), q.push(t);
64     long long distl = dist(x, lc[mid]), distr = dist(x, rc[mid]);
65     if (distl > q.top() && distr > q.top()) {
66         if (distl > distr) {
67             query(l, mid - 1, x);
68             if (distr > q.top()) query(mid + 1, r, x);
69         } else {
70             query(mid + 1, r, x);
71             if (distl > q.top()) query(l, mid - 1, x);
72         }
73     } else {
74         if (distl > q.top()) query(l, mid - 1, x);
75         if (distr > q.top()) query(mid + 1, r, x);
76     }
77 }
78
79 int main() {
80     cin >> n >> k;
81     k *= 2;
82     for (int i = 1; i <= k; i++) q.push(0);
83     for (int i = 1; i <= n; i++) cin >> s[i].x >> s[i].y;
84     build(1, n);
85     for (int i = 1; i <= n; i++) query(1, n, i);
86     cout << q.top() << endl;
87     return 0;
88 }

```

## 习题

[「SDOI2010」捉迷藏](#)


[「Violet」天使玩偶/SJY 摆棋子](#)

[「国家集训队」JZPFAR](#)

[「BOI2007」Mokia 摩基亚](#)

 本页面最近更新：2023/12/16 12:24:12, [更新历史](#)

 发现错误？想一起完善？ [在 GitHub 上编辑此页！](#)

 本页面贡献者：[hsfzLZH1](#), [Ir1d](#), [AC-Stray](#), [CCXXI](#), [dkz051](#), [Eletary](#), [Enter-tainer](#), [Henry-ZHR](#), [kenlig](#), [minamimelon](#), [ouuan](#), [Rainboylvx](#), [shuzhouliu](#), [StudyingFather](#), [Tiphereth-A](#), [Xeonacid](#)

© 本页面的全部内容 [在 CC BY-SA 4.0 和 SATA 协议之条款下提供](#)，附加条款亦可能应用

1 个表情



17 条评论 · 3 条回复 - 由 *giscus* 提供支持

最早

最新

输入

预览

Aa

登录后可发表评论

使用 GitHub 登录

morris821028 2020 年 2 月 4 日

補充：複雜度分析上，用 `sort` 的確是比較慢的，但是對於遞迴實作時，我們若採用相同的陣列，則劃分時容易造成部分排序好，此時時間複雜度略低於 `nth_element` 的。

而使用 `nth_element` 時，不容易實作出打破等價的代碼，這使得左右子樹皆可能涵蓋中間值，對於需要剪枝的算法可能是一個效能退化的點。

↑ 1 1

0 条回复

shrtcl 2020 年 4 月 15 日

最近点对那题 `build` 函数 `l>=r` 应该改成 `l>r` 吧，不然叶子结点有可能没计算到，那题数据比较水

↑ 1

0 条回复

shrtcl 2020 年 4 月 15 日

说错了当我没说

↑ 1

0 条回复



skip2004 2020 年 5 月 21 日

kdt重构不应该使用替罪羊树的方法，因为kdt复杂度可以粗略表示为 $\sqrt{2}^{\text{dep}}$ ，而替罪羊树的深度是 $O(\log n)$ 而不是 $\log n$ ，最好的方法是隔约 $\sqrt{n}$ 次操作后重构。当然如果可以离线可以先离线操作建树，这样就避免了插入。

↑ 1 

0 条回复

morris821028 2020 年 5 月 21 日

@skip2004

kdt重构不应该使用替罪羊树的方法，因为kdt复杂度可以粗略表示为 $\sqrt{2}^{\text{dep}}$ ，而替罪羊树的深度是 $O(\log n)$ 而不是 $\log n$ ，最好的方法是隔约 $\sqrt{n}$ 次操作后重构。

替罪羊树为均摊操作，某一次可能跑比较慢达到整棵树重建，基本上是不影响整体的需求。而你提供的  $\sqrt{n}$  次操作就重构是局部子树还是整棵树？如果不能保证询问和插入的数量比例的趨勢，很難給予一個最佳的分析，這可能造成卡在臨界  $\sqrt{n}$  的操作前，不斷地詢問造成單次操作  $O(\sqrt{n})$

↑ 1 

0 条回复

skip2004 2020 年 5 月 21 日

@morris821028 我的方法是隔  $\sqrt{n}$  次操作重构整棵树，可以证明总复杂度为  $n\sqrt{n}\log n$ 。

然而如果你直接使用替罪羊树，在比较强力的数据之下，询问时可以将复杂度卡至  $\sqrt{2}^{\text{dep}}$ ，而替罪羊树的深度是可以卡到比较大的，比如  $1.5\log n$ ，这样子复杂度就不再是  $\sqrt{n}$  了，将会是更大。

↑ 1 

0 条回复

morris821028 2020 年 5 月 21 日

已编辑

@skip2004

替罪羊深度最多为  $O(\log n)$  的，而你每  $\sqrt{n}$  才调整一次，那深度应该就会达到  $\sqrt{n}$ 。套上相同的 kd tree 搜索逻辑。如果询问复杂度是深度的一个函数，要做哪一种询问才会到你所说的单次  $\sqrt{2}^{\text{dep}}$ ，而每  $\sqrt{n}$  次调整的确不会遇到这个问题？

↑ 1 

0 条回复

skip2004 2020 年 5 月 21 日

@morris821028

@skip2004

替罪羊深度最多為  $O(\log n)$  的，而你每  $\sqrt{n}$  才調整一次，那深度應該就會達到  $\sqrt{n}$ 。套上相同的 kd tree 搜尋邏輯。如果詢問複雜度是深度的一個函數，要做哪一種詢問才會到你所說的單次  $\sqrt{2}^{\text{dep}}$ ，而每  $\sqrt{n}$  次調整的確不會遇到這個問題？

这是一个特殊情况，我们在搜索时，除了新加的点外的复杂度最多是  $\sqrt{n}$  的，而访问每个新加的点最多花费  $\log(n)$  的时间，所以新加的点产生的额外搜索时间不会超过  $\sqrt{n}\log(n)$ ，总的来说总复杂度也不会超过  $\sqrt{n}\log(n)$

↑ 1 

0 条回复

Yoshinow2001 2021 年 2 月 27 日

@shrtcl

说错了当我没说

话说我现在也有相同的疑问，为什么  $l=r$  的时候 return 0 也可以哇orz

↑ 1 

0 条回复

myeeye 2021 年 11 月 4 日

似乎可以考虑加入二进制分组等实际复杂度更优的重构策略？

↑ 1 

0 条回复

Cat-shao 2022 年 1 月 24 日

第三十一行 `if (l >= r) return 0;` 是不是应该为 `l > r`。  
改成 `l > r` 后照样能 AC。

↑ 1 

0 条回复

xiaohaowudi 2022 年 1 月 27 日

luogu P1429 半圆最近点对 (加强版) 代码貌似与错了吧? build时候 返回0条件应该是  $l > r$ , 不是  $l \geq r$ , OJ上数据水没有测出来

↑ 1 

0 条回复

Hanghang007 2023 年 2 月 11 日

话说现在存在有题卡替罪羊重构然后放根号重构麻?

有题卡轮换切割放方差切割的麻?

我认为能用kdt解决且k都是一样的, 然后加上了剪枝的代码基本都没有题目卡吧。

↑ 1 

1 条回复

JohnAlfnov 2023 年 6 月 24 日

P4357 K远点对 kdt过不去了



cycyyds 2023 年 7 月 10 日

这个类似替罪羊树的动态加点时间复杂度是假的吧

lxl 的 PPT 图片镇楼:

[https://cdn.luogu.com.cn/upload/image\\_hosting/fxc1qrce.png?x-oss-process=image/resize,m\\_lfit,h\\_170,w\\_225](https://cdn.luogu.com.cn/upload/image_hosting/fxc1qrce.png?x-oss-process=image/resize,m_lfit,h_170,w_225)

↑ 1 

1 条回复

clx0226 2023 年 8 月 21 日

捕捉一只cyc,但你这清晰度.....



hhc0001 2023 年 7 月 20 日

那如果真的把KDT和平衡树结合到一起又会怎么样呢?

↑ 1 

1 条回复

hhc0001 2023 年 7 月 20 日

毕竟KDT有很多的性质都源于二叉搜索树（平衡树）？



qiyuewuyi2333 6月11日

为什么构建k-D Tree时，要轮换使用的轴？

You just divide the space by x-axis. But now you have to find those points in the area given by y, such as -1. In this case, the k-D Tree you built thoroughly loses efficacy.

Maybe this case is a little extreme, but it could give us one inspiration: If you just build a k-d Tree by one axis, then you lose the infos in the other dimensions. So we need to alternate the splitting dimension by turns. That also makes the value of info of each dimension balanced.

But you also could say I just use an x-axis to search. Yeah, in this situation, you certainly could just use x-axis to build the tree. But this obviously makes the k-d tree **degenerate** to balanced search tree. It's no sense. Please think about the reason you