基于 k-d 树的查询算法实现与二维可视化

王怡力
计研三一
2020310829

郑欣阳
高等研究院
2020311848

1 问题描述

k-d 树 (k-dimensional tree) 是由斯坦福大学的 Bentley 教授 [1] 在 1975 年提出的一种在 K 维空间中对数据集进行层次划分的数据索引结构, 它不仅服务于计算几何领域，而且在计算机图形学，地理信息系统，统计分析以及机器学习领域都有广泛应用。

在本次计算几何大作业中，我们实现了基于 k-d 树的半径查询（范围查询）、近邻查询 (KNN) 共四种空间查询算法以及它们在二维情况下的可视化。

2 数据结构

标准 k-d 树是一棵二叉树，假设我们要存储的数据记为 n 个容量为 k 的键, 那么 k-d 树的每个结点都对应于被某个键 (`key`) 和分割维 (`discriminant`) $\in [0, k-1]$ 定义的一个超平面。初始时, 根节点对应整个空间, 我们令 $\mathbf{x}$ 作为根节点的键, i 作为它的分割维. 那么, 整个根节点空间被 $\mathbf{x}[i]$ 划分为了两个子树空间: 所有满足 $\mathbf{y}[i] < \mathbf{x}[i]$ 的键被归入左子树, 所有满足 $\mathbf{y}[i] \geq \mathbf{x}[i]$ 的键被归入右子树, 同样的划分递归地在每个子树上执行, 直到划分出的树为空时结束. 分割维按顺序从 0 开始轮流遍历, 即深度为 d 的结点的分割维为 $d \bmod k$。

3 算法说明

本部分阐述了 k-d 树的构造方法, 并以范围查询和近邻查询算法为代表, 说明空间查询算法的原理。

3.1 k-d 树的构建与插入

Algorithm 1 实时 kd 树的插入

```
procedure INSERT(T, x, v)
    if T = □ then INSERTELEMENT(x, v)
    key ← T.key; i ← T.discr
    if x[i] < key[i] then INSERT(T.left,x, v)
    else INSERT(T.right, x, v)
end procedure
```

对于离线的数据, 我们能选择分割维的中点作为分割点构造出有利于降低查询算法复杂度的平衡 k-d 树, 对于实时的数据, 我们需要执行插入算法1构造标准 k-d 树. 以二维情况为例, 假设一组数据为

$(6,4), (5,2), (4,7), (8,6), (2,1), (9,3)$ and $(2,8)$ 作为离线数据和依次插入的实时数据, 构建出的 k-d 树结构如图1和图2所示:

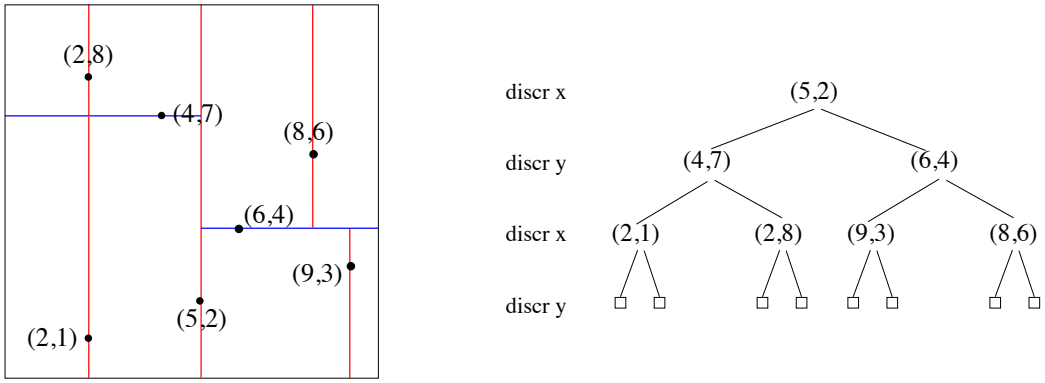

Figure 1: offline 2d-tree structure

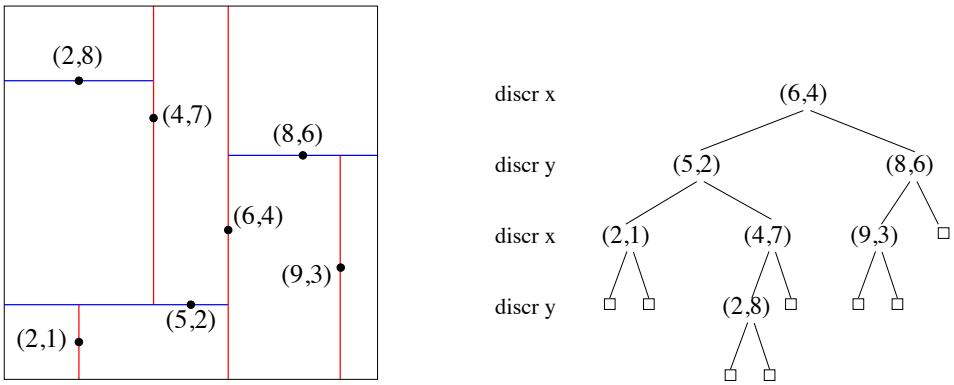

Figure 2: online 2d-tree structure

3.2 基于 k-d 树的最近邻与 k-近邻查询

最近邻查询与半径查询（范围查询）不同, 它没有固定的查询路径. 算法在查询时需要维护与搜索点最近的叶子的距离, 通过比较搜索点到子树距离的下界与当前最小距离来进行动态的剪枝 (`prune`)[2]。

k-近邻与最近邻的整体思路并没有大的区别，增加的部分是，k-近邻查询需要维护一个优先队列存储最优的 k 个节点，这样每次比对回溯节点是否比当前最优点更优的时候，就只需用当前最优点中里 k 最远的节点来比对，而这个工作对于优先队列来说是 $\mathcal{O}(1)$ 的。

3.3 基于 k-d 树的半径范围查询

在二维欧式空间, 半径范围查询的返回对象是满足条件 $d(c,p) \leq r$ 的所有点 p, 其中中心点为 c, 半径为 r, 算法表示为:

其中 ComputeBoundingBoxes(...) 计算出左子树和右子树对应的包围盒，左右两个包围盒被 Intersects(BB, c, r) 调用和目标圆求交，我们在这里用目标圆的包围盒近似实现. 图3是一个由半径 1.5 和中心点 $(3,7)$ 定义的半径范围查询, 在这个例子里, 算法返回 $(4,7)$ 和 $(2,8)$。

Algorithm 2 半径范围查询

procedure RADIUSRANGE(T, BB, c, r, L)
  if $T = \square$ **then return**
  $key \leftarrow T.key$; $i \leftarrow T.discr$
  if DISTANCE(key, c) $\leq r$ **then** $L \leftarrow L \cup \{key\}$
  COMPUTEBOUNDINGBOXES($lBB, rBB, BB, key[i], i$)
  if INTERSECTS(lBB, c, r) **then**
    RADIUSRANGE($T.left, lBB, center, radius, L$)
  if INTERSECTS(rBB, c, r) **then**
    RADIUSRANGE($T.right, rBB, center, radius, L$)
end procedure

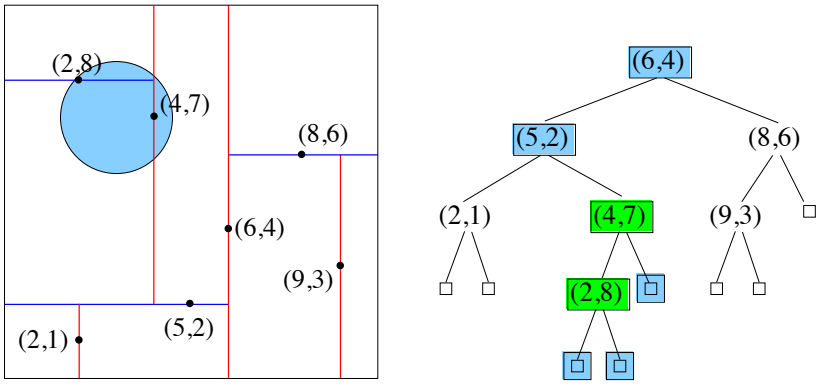

Figure 3: Randius Range in a 2d-tree using the Euclidean distance

3.4 基于 k-d 树的正交范围查询

查询返回对象是给定超矩形内的所有数据点, 超矩形由最上边界 ($lowerBound$) 和最下边界 ($upperBound$) 定义, 算法表示为:

Algorithm 3 正交范围查询

procedure ORTHOGONALRANGE($T, lowerBound, upperBound, L$)
  if $T = \square$ **then return**
  $key \leftarrow T.key$; $i \leftarrow T.discr$
  if INSIDE($key, lowerBound, upperBound$) **then** $L \leftarrow L \cup \{key\}$
  if $lowerBound[i] < x[i]$ **then**
    ORTHONGONALRANGE($T.left, lowerBound, upperBound, L$)
  if $upperBound[i] \geq x[i]$ **then**
    ORTHONGONALRANGE($T.right, lowerBound, upperBound, L$)
end procedure

其中 INSIDE($key, lowerBound, upperBound$) 是一个用来确定键是否位于矩形内的函数. 以上文的实时 k-d 树为例, 图4是一个由 $lowBound = (1, 5)$ 和 $upperBound = (5, 9)$ 定义的正交范围查询, 在这个例子里,$(4, 7)$ 和 $(2, 8)$ 在矩形内部。

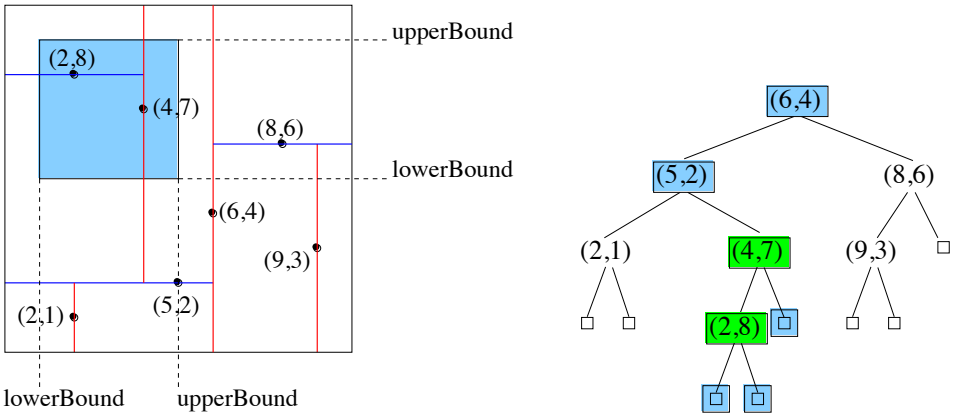

Figure 4: Orthongonal Range Search in a 2d-tree

4 程序性能分析

我们考虑不同的情境设置对程序性能的影响，对程序的性能进行分析。

4.1 建树耗时

我们可以考虑点集遵循不同的数据分布，即随机分布 (random 分布)、落在椭圆环上 (circle 分布)、落在对角线上 (diagonal 分布)，得到的结果如图5所示：

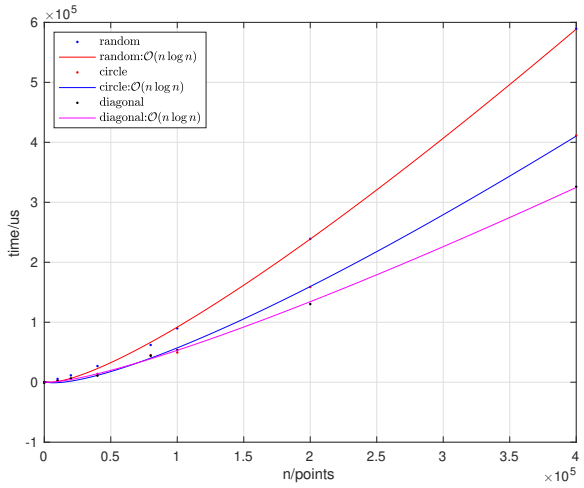

Figure 5: 不同的数据分布下的建树耗时

理论上建树的时间复杂度为 $\mathcal{O}(n \log n)$，从图5中可以看到，实验结果与理论估计吻合得很好。注意到对于依 random 分布的数据，程序表现最差，原因可能是由于依 random 分布的数据相比于另外两种分布没有特定的结构。

4.2 k-近邻查询耗时

对于 k-近邻查询，我们也可以考虑点集遵循不同的数据分布，即依 random 分布、circle 分布、diagonal 分布，我们假设查询点的分布为随机分布，于是可以随机取 1000 个查询点计算耗时的平均值当作查询的耗时，固定建树的点 $n = 10000$，分析 k-近邻查询耗时与 k 的关系，得到的结果如图6所示：

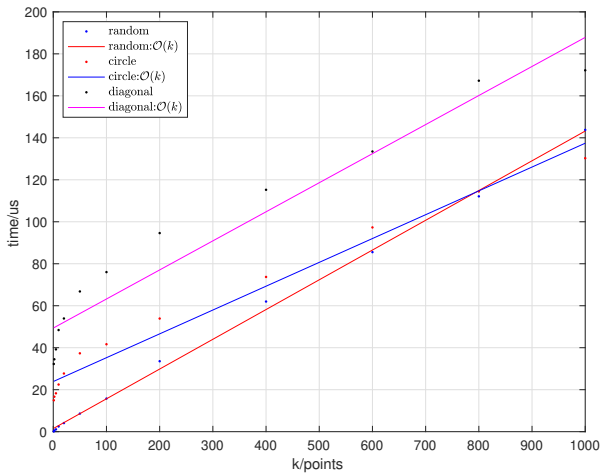

Figure 6: $n = 10000$，不同的数据分布下的 k-近邻查询耗时与 k 的关系

在固定 n 的情况下，理论上 k-近邻查询耗时与 k 应当成线性关系，但实验表明只有依 random 分布的数据比较好地展现了线性地规律。为什么依 cicle 分布和依 diagonal 分布的数据没有展现这一规律呢？这里的原因也很简单，考虑如图7和图8所示的情况，注意到对于具有特定结构的分布，在某些查询点处可能需要访问 k-d 树的较低层的子树才能完成一次查询，所以 k-近邻查询耗时还受数据分布影响。

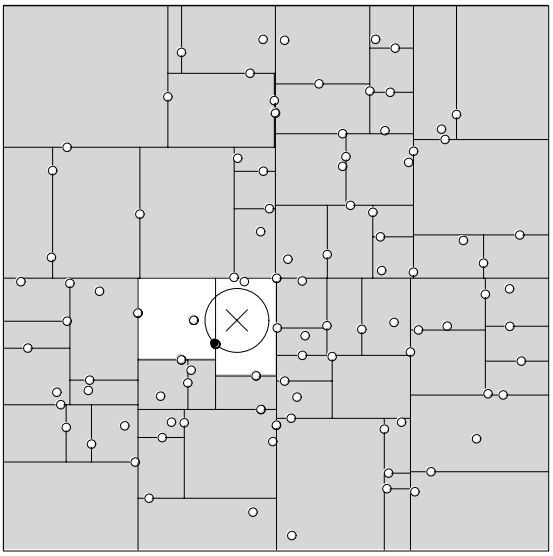

Figure 7: 耗时较少的情况

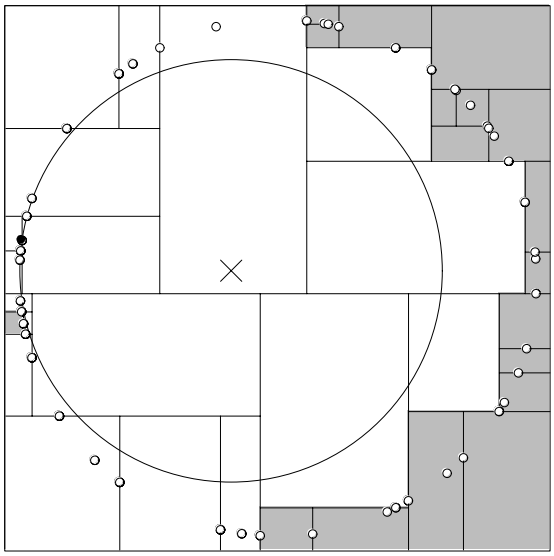

Figure 8: 耗时较多的情况

固定 k-近邻 $k = 20$，分析 k-近邻查询耗时与 n 的关系，得到的结果如图9所示：

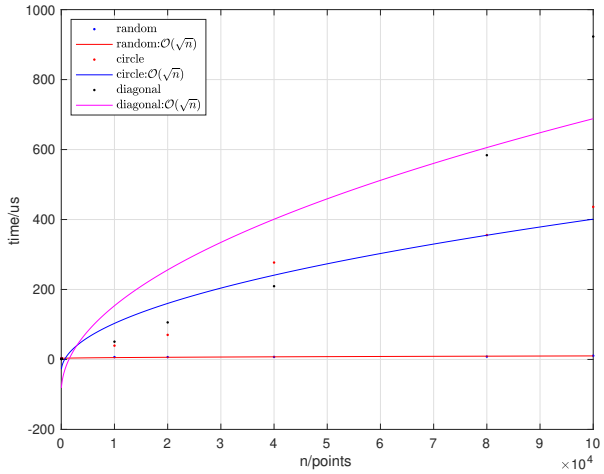

Figure 9: $k = 20$，不同的数据分布下的 k-近邻查询耗时与 n 的关系

在固定 k 的情况下，理论上 k-近邻查询的时间复杂度为 $\mathcal{O}(\sqrt{n})$，但实验表明只有依 random 分布的数据比较好地展现了这一规律。对于依 cicle 分布和依 diagonal 分布的数据，原因同上，k-近邻查询耗时还受数据分布的影响，且这种影响可能会盖过 $\mathcal{O}(\sqrt{n})$ 的影响，进一步地，通过图9，我们看到数据分布也会极其显著地影响 k-近邻查询的效率。

4.3 半径范围查询耗时

对于半径范围查询，我们同样考虑点集遵循不同的数据分布，即依 random 分布、circle 分布、diagonal 分布，假设查询点的分布为随机分布，同样随机取 1000 个查询点计算耗时的平均值当作查询的耗时，固定建树的点 $n = 10000$，分析半径范围耗时与 r 的关系，得到的结果如图10所示，其中 width 和 height 规定了点 (x, y) 的分布区域，即 $x \in [0, \text{width}], y \in [0, \text{height}]$。

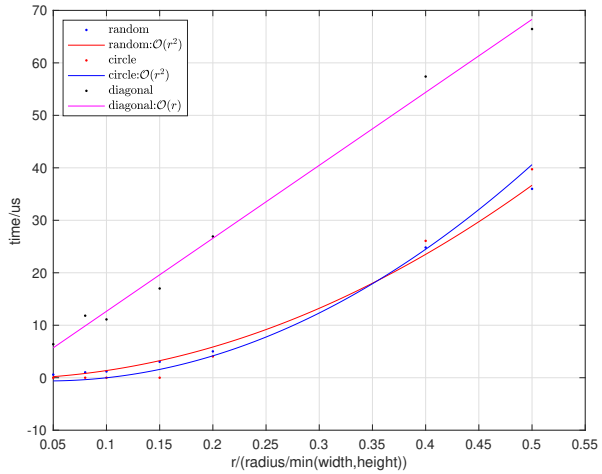

Figure 10: $n = 10000$，不同的数据分布下的半径范围查询耗时与 r 的关系

我们知道，对于随机分布的查询点，在期望意义下，依 diagonal 分布情况下所能查询到的点与 r 成正比，而依 random 分布和依 circle 分布情况下所能查询到的点与 r^2 成正比，故依 diagonal 分布情况下的时间复杂度为 $\mathcal{O}(r)$，而依 random 分布和依 circle 分布情况下的时间复杂度为 $\mathcal{O}(r^2)$，从图10中可以看到，实验结果与理论估计吻合得很好。

考虑固定半径 $r = 0.2\min(\text{width}, \text{height})$，分析半径范围查询耗时与 n 的关系，得到的结果如图11所示，

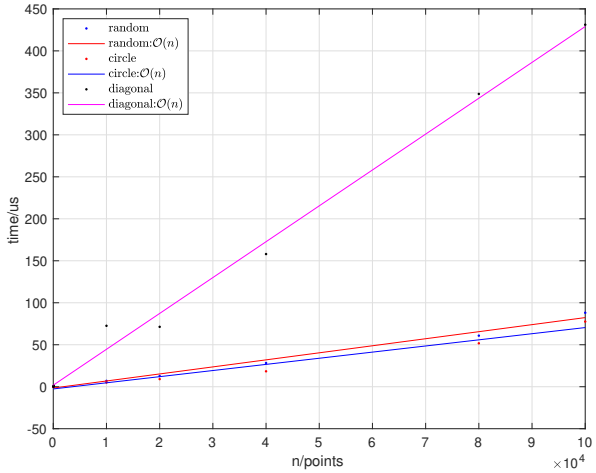

Figure 11: $r = 0.2\min(\text{width}, \text{height})$，不同的数据分布下的半径范围查询耗时与 n 的关系

当固定 r 的时候，由于 n 越大，所需要报告的点数就越多，不同的点集分布下的半径范围查询耗时在期望意义下与 n 近似成线性关系，图11也验证了这一结论。

4.4 Acknowledgement

测试程序性能所使用的处理器为 2.7 GHz 四核 Intel Core i7，内存为 16 GB 2133 MHz LPDDR3，图5中的曲线拟合使用了 Matlab 的 cftool 工具箱，测试所得的具体数据可见附录。

5 小组分工

本项目文件的主要内容没有用在其它课程作为课程项目或者作业提交，两位同学对本项目有同等的贡献。

6 实验总结

在本次计算几何大作业中，我们实现了基于 k-d 树的半径查询（范围查询）、近邻查询 (KNN) 共四种空间查询算法以及它们在二维情况下的可视化，并对 k-d 树在不同的情境下的性能做了分析与讨论。

References

[1] Jon Louis Bentley. “Multidimensional Binary Search Trees Used for Associative Searching”. In: *Commun. ACM* 18.9 (Sept. 1975), pp. 509–517.

[2] Carl Kingsford. *kd-Trees*. https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/kdtrees.pdf. Accessed: 2021–06-17. 2008.

7 附录

依随机分布 (random 分布)、落在椭圆环上 (circle 分布)、落在对角线上 (diagonal 分布) 的点分别如下图所示（图中各有 10000 个点）：

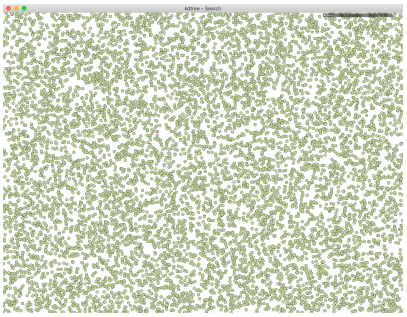

Figure 12: random 分布

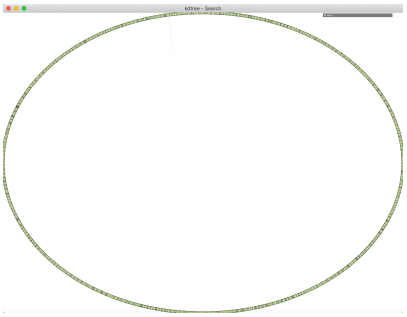

Figure 13: circle 分布

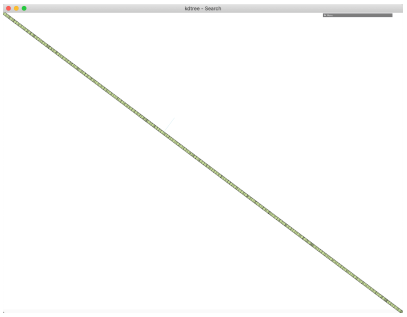

Figure 14: diagonal 分布

7.1 建树耗时

7.1.1 random 分布

对于 random 分布，我们有测试结果：

n/points	1	5	25	50	100	10000	20000	40000	80000	100000	200000	400000
time/μs	0	1	3	8	19	5494	11429	26944	62065	89771	238990	589308

Table 1: random 分布下建树的耗时

7.1.2 circle 分布

对于 circle 分布，我们有测试结果：

n/points	1	5	25	50	100	10000	20000	40000	80000	100000	200000	400000
time/μs	0	2	5	9	20	2372	5395	13771	45230	49534	158761	411274

Table 2: circle 分布下建树的耗时

7.1.3 diagonal 分布

对于 diagonal 分布，我们有测试结果：

n/points	1	5	25	50	100	10000	20000	40000	80000	100000	200000	400000
time/μs	0	1	3	7	13	2199	6306	10942	43981	53568	130291	325887

Table 3: diagonal 分布下建树的耗时

7.2 k-近邻查询耗时

7.2.1 random 分布

当 $n = 10000$ 时，考虑 k 的影响，我们有测试结果：

k/points	1	2	5	10	20	50
time/μs	0.126	0.542	1.102	2.479	4.097	8.603
k/points	100	200	400	600	800	1000
time/μs	15.769	33.558	61.997	85.53	112.107	143.769

Table 4: $n = 10000$，random 分布下 k-近邻查询耗时与 k 的关系

当 $k = 20$ 时，考虑 n 的影响，我们有测试结果：

n/points	25	50	100	10000	20000	40000	80000	100000
time/μs	2.799	3.103	2.547	6.675	6.632	7.308	8.185	10.641

Table 5: $k = 20$，random 分布下 k-近邻查询耗时与 n 的关系

7.2.2 circle 分布

当 $n = 10000$ 时，考虑 k 的影响，我们有测试结果：

k/points	1	2	5	10	20	50
time/μs	14.944	16.657	18.268	22.456	27.708	37.328
k/points	100	200	400	600	800	1000
time/μs	41.65	53.897	73.713	97.302	114.426	130.282

Table 6: $n = 10000$，circle 分布下 k-近邻查询耗时与 k 的关系

当 $k = 20$ 时，考虑 n 的影响，我们有测试结果：

n/points	25	50	100	10000	20000	40000	80000	100000
time/μs	3.331	2.323	2.439	39.444	70.209	277.005	355.439	436.19

Table 7: $k = 20$，circle 分布下 k-近邻查询耗时与 n 的关系

7.2.3 diagonal 分布

当 $n = 10000$ 时，考虑 k 的影响，我们有测试结果：

k/points	1	2	5	10	20	50
time/μs	32.258	34.474	39.208	48.409	53.925	66.801
k/points	100	200	400	600	800	1000
time/μs	76.019	94.605	115.232	133.481	167.199	172.157

Table 8: $n = 10000$，diagonal 分布下 k-近邻查询耗时与 k 的关系

当 $k = 20$ 时，考虑 n 的影响，我们有测试结果：

n/points	25	50	100	10000	20000	40000	80000	100000
time/μs	1.01	1.405	1.913	50.804	105.85	209.342	583.803	923.039

Table 9: $k = 20$，diagonal 分布下 k-近邻查询耗时与 n 的关系

7.3 半径范围查询耗时

7.3.1 random 分布

当 $n = 10000$ 时，考虑 r 的影响，我们有测试结果：

r/(radius/min(width, height))	0.05	0.08	0.1	0.15	0.2	0.4	0.5
time/μs	0.595	1.032	1.189	3.036	5.019	24.815	35.989

Table 10: $n = 10000$，random 分布下半径范围查询耗时与 r 的关系

当 $r = 0.2 \min(\text{width}, \text{height})$，考虑 n 的影响，我们由此测试结果：

n/points	25	50	100	10000	20000	40000	80000	100000
time/μs	0	0.108	0.418	6.977	12.748	28.227	60.829	88.098

Table 11: $r = 0.2 \min(\text{width}, \text{height})$，random 分布下半径范围查询耗时与 n 的关系

7.3.2 circle 分布

当 $n = 10000$ 时，考虑 r 的影响，我们有测试结果：

r/(radius/min(width, height))	0.05	0.08	0.1	0.15	0.2	0.4	0.5
time/μs	0	0	0.001	0.001	4.052	26.08	39.729

Table 12: $n = 10000$，circle 分布下半径范围查询耗时与 r 的关系

当 $r = 0.2 \min(\text{width}, \text{height})$，考虑 n 的影响，我们由此测试结果：

n/points	25	50	100	10000	20000	40000	80000	100000
time/μs	0	0	0	5.006	9.083	18.355	51.701	77.601

Table 13: $r = 0.2 \min(\text{width}, \text{height})$，circle 分布下半径范围查询耗时与 n 的关系

7.3.3 diagonal 分布

当 $n = 10000$ 时，考虑 r 的影响，我们有测试结果：

r/(radius/min(width, height))	0.05	0.08	0.1	0.15	0.2	0.4	0.5
time/μs	6.382	11.822	11.112	17.005	26.915	57.394	66.45

Table 14: $n = 10000$，diagonal 分布下半径范围查询耗时与 r 的关系

当 $r = 0.2 \min(\text{width}, \text{height})$，考虑 n 的影响，我们由此测试结果

n/points	25	50	100	10000	20000	40000	80000	100000
time/μs	0.014	0.426	1.237	72.563	71.327	158.002	348.711	431.176

Table 15: $r = 0.2 \min(\text{width}, \text{height})$，diagonal 分布下半径范围查询耗时与 n 的关系