最近邻搜索实验报告

本次实验我尝试了 8 种不同的算法：
1.线性搜索；2.KD 树；3.球树；4.优先队列；5.快速选择；6.LSH；7.HNSW;
除了以上的算法，我还尝试了一个自己的算法，不过并没有用。。。。。

该实验报告将会介绍每一个我尝试的算法以及其中的实现过程，其中 HNSW 和 LSH 由于实现难度大(查了一个周末的资料都不是很明白。。。)，便交由 gpt 实现。

所有的代码及其召回率均在压缩包里。

本地实验环境：
CPU：AMD Ryzen 5 7535U with Radeon Graphics;
内存：16.0 GB (15.2 GB 可用);
操作系统：win11;
编程语言：C++;
编译器：visual studio2022;
实验控制的变量：
均使用 cin 和 cout 进行输出，无任何 io 优化;
均使用单核编译;
使用 testcase 和 sift 两个数据集进行测试(testcase 的维度为 8, k 值为 30, sift 的维度为 128, k 值为 100);
Testcase 的测试时间上限：10 分钟;
Sift 的测试时间上限：8 小时;
测试的数据包括测试时间和算法的召回率;
测试时间只包含查询时间和输出时间，不包含任何前置时间(一些初始化的过程)

实验记录和结果：
1. 线性搜索：
思路：
计算查询向量于数据集中的每一个向量，再将这些结果进行排序，输出前 k 个;
时间复杂度：O(nd);
空间复杂度：O(1);

算法评价与概述：
该算法是最慢的一个算法，也最直接，eoj 的案例通过数如下：

提交 #3649287

#	送交者	题目	语言	提交时间	评测时间	结果	耗时	内存	评测机
3649287	10235101477	A. 相似最近邻搜索	C++17	2024-05-04 16:55:04	2024-05-04 16:55:32	Time limit exceeded: 7	2.013	2.551	Atlanda
	✗ ✗ ✗ ✗ ✗ ✗ ✗ ✓ ✗ ✗ ✗ ✗ ✗ ✗								

由于在其测试 testcase 的时间超过了 10 分钟，便索性不再测试 sift 数据集。

2. 我自己的算法：

思路：

按照数据集中的向量到原点的距离进行排序，对于每一个查询向量，根据其到原点的距离找到数据集中与原点距离差不多的点，进行临近搜索。

时间复杂度：O(k);//以离原点距离差不多的点作为中心，临近搜索的范围为-k~k。

空间复杂度：O(k);

算法评价与概述：

该算法属于 ANN，实际操作过程中该算法的召回率过低.

该算法的 eoj 尝试：

提交 #3649308

#	送交者	题目	语言	提交时间	评测时间	结果	耗时	内存	评测机
3649308	10235101477	A. 相似最近邻搜索	C++17	2024-05-04 18:56:11	2024-05-04 18:56:23	Wrong answer: 7	1.202	214.844	Atlanda

Testcase 数据跑分结果：

```
//自己的算法
第一次：27.196s；第二次：26.980s；第三次：26.358s；
```

该算法由于召回率较低(0~40%,少量达到 60%，大部分为 0)，便没有测试 sift 数据集。

3. KD 树：

思路：

利用数据的维数将数据存到一个二叉树中，插入规则为：先以第一个的数据的第一维作为标准，之后数据的第一维大于这个数据的第一维的，插入左子树，否则插入右子树，对于这两个子树，插入的规则就变为了比较第二维的数据，以此类推，到最后一维时，下一次又从第一维开始进行比较，直到插满数据。对于每一个查询向量，先按插入规则走到叶子节点，在回溯搜索其附近的节点。

时间复杂度：平均：O(logn)；最坏：O(n);

空间复杂度：O(nd);

算法评价与概述：

该算法是我通过 gpt 问到的一个算法，属于 ANN 算法，我通过自学 b 站上的视频([JHML-KNN-03]knn 算法-kd 树原理(构造)_哔哩哔哩_bilibili)自己学会构建 KD 树，然后利用 gpt 和视频中的构造案例验证了自己构造的代码的正确性，接着在 eoj 上提交：

提交 #3649459

#	送交者	题目	语言	提交时间	评测时间	结果	耗时	内存	评测机
3649459	10235101477	A. 相似最近邻搜索	C++17	2024-05-05 11:29:06	2024-05-05 11:29:23	Wrong answer: 21	2.006	7.066	Atlanda

只过了 3 个用例，相比于前面两个好了不少。
以下是 testcase 的数据测试：

```
* //KD树
*第一次：12.123s；第二次：12.245s;第三次：12.289s;
```

这个速度相当快，但是其召回率却比较低（40%~60%）
以下是 sift 的数据测试：

```
*第一次：18.526s；第二次：19.183s;
```

召回率：小于 10%
Sift 和 testcase 的数据差了 100 多倍(sift 数据是 1M，testcase 是 8000)，速度却没差多少，这是一个令人吃惊的结果，思考原因，可能是二叉树的深度再 1M 的数据下也仅仅 20 出头，对于一个数据的比较并没有多多少次，不过仍有疑点，sift 数据是 128 维的，testcase 是 8 维的，查找时的计算次数应该多更多，但是此处我并没有进一步地实验，是由于其召回率从 40%~60%降到了小于 10%，再根据 gpt 提供的信息，kd 树只对 20 维以下的数据效率较高，便没有深究。

4. 球树：

思路：
与 kd 树类似，球树是一个二叉树，球树通过找到一个尽可能小的球将所有的数据包住，接着根据这个数据集的数据到数据集中最远的两个点的距离，将数据集划分为两个部分，这两部分重复上述的过程（每个部分独立建球），直到这个球中数据量小于或等于 k 值，这样就完成了球树的构建。对于每一个查询向量，通过计算与球树中子球球心的距离，与最初计算与一级左子球和一级右子球的距离进行比较判断是否进入该子树进行查询，进入最终的子树后再搜集结果。
时间复杂度：O(logn);
空间复杂度：O(nd);
算法评价与概述：
该算法也是属于 ANN，我是通过 gpt 得知该算法。该算法有用的中文资料少得可怜，b 站上也没有相关构建球树的视频，我所构建的球树参考了一个知乎的博主（舟晓南 - 知乎 (zhihu.com)），我利用该博主提供的用例和 eoj 的用例测试了我写的球树，均构建成功，接着我提交了 eoj：

提交 #3671734

#	送交者	题目	语言	提交时间	评测时间	结果	耗时	内存	评测机
3671734	10235101477	A. 相似最近邻搜索	C++17	2024-05-26 14:56:29	2024-05-26 14:56:56	Time limit exceeded: 14	2.005	4.461	Atlanda

只过了两个。
Testcase 的数据测试：

//球树
第一次：355.021s；第二次：371.424s；第三次：363.989s

整体速度较比后面的优先队列和快速选择要慢，再看一下 sift 的数据：

第一次：29176.174s;第二次：21700.601s

都超过了 6 个小时，值得注意的是，两次数据的结果相差较远，经过分析，发现可能是因为第一次测试时是挂后台跑的，而第二次是挂前台跑的，后面的优先队列和快速选择可以证明挂后台和挂前台的速度差别(也是测了两次，一次挂前台，一次挂后台)。

5. 优先队列：

思路：

计算查询向量与每一个向量的距离，在计算的过程中，维护一个大根堆，若查询向量与该向量的距离小于这个大根堆的堆顶元素，那么将堆顶元素换为该向量与查询向量的距离，接着调整堆使其仍为大根堆，大根堆的大小由 k 值确定（因为我们只要前 n 个即可）。

时间复杂度：O(nlogk);

空间复杂度：O(k);

算法评价与概述：

该算法属于 KNN，精确查找，该算法是从经过王丽苹老师的提示后上网查资料得知，优先队列是 topk 问题的一个热门解法，之前一直查的是 ANN，而并没有查过 KNN(一直以为只有 ANN 能过。。。)。

在刚开始写这个算法的时候，我以为是将整个数据集建堆，但是在 eoj 上尝试后发现效率并不高，后来又观看了 b 站的相关视频（面试官问一亿个数据怎么取前十榜单，这种 top k 问题你只会答先排序再取前十？今天一次性教会你堆排序与 top K 原理与实现学会后直接甩面试官脸上！（一）_哔哩哔哩_bilibili），才明白是维护一个大根堆进行操作，值得一提的是，尽管中间走了弯路，写这个算法的时间还是比写球树和写 kd 树的时间要快(球树写了一天，kd 树花了一个上午，而该算法只花了两小时的时间写出)，以下是 eoj 上尝试的提交：

提交 #3651471

#	送交者	题目	语言	提交时间	评测时间	结果	耗时	内存	评测机
3651471	10235101477	A. 相似最近邻搜索	C++17	2024-05-08 18:19:38	2024-05-08 18:20:06	Time limit exceeded: 14	2.013	2.625	Atlanda
	✔ ✖ ✖ ✖ ✖ ✖ ✖ ✔ ✖ ✖ ✖ ✖ ✖								

这个是将整个数据集建堆的版本。

提交 #3651503

#	送交者	题目	语言	提交时间	评测时间	结果	耗时	内存	评测机
3651503	10235101477	A. 相似最近邻搜索	C++17	2024-05-08 19:31:45	2024-05-08 19:32:04	Time limit exceeded: 64	2.004	5.898	Atlanda

需要帮助吗?

C++17

这个是真正的优先队列的版本，也不是第一次就写好的，后面还有一些尝试。

提交 #3651536

#	送交者	题目	语言	提交时间	评测时间	结果	耗时	内存	评测机
3651536	10235101477	A. 相似最近邻搜索	C++17	2024-05-08 19:58:14	2024-05-08 19:58:31	Time limit exceeded: 79	2.001	6.941	Atlanda

最终分数为 79 分。

不过优先队列的极限不是 79 分，是满分（以下是卡常过程，可跳过）：
在 79 分之后，我一直以为 LSH 是这道题的正解，直到我在尝试了 LSH 后发现根本写不出来一个好的 LSH，抱着试一试的心态，我尝试将优先队列遍历的初始位置从 0 改为 n*0.05(想着 80%的召回率应该很高？)，当时以为这样的改动是不行的，但是，提交后之前过的用例都过了，接着我将 n 前的这个系数进一步提高，花了 20 分钟左右的时间将这个系数确定为 0.198(0.199 就会有 WA 了)，我对比了之前跑通的数据，大概快了 0.3s，但分数还是 79，接着我又尝试把快读快写加入，不过当时并不知道数据的输入格式中有按科学计数法的输入，快读的代码是失效的，只能加入快写的代码，巧合的是，那天下午，有同学利用卡常将这道题 ac 了(没有这个同学，我估计会放弃卡常)，我到当天晚上还没有超过 79 分，回到宿舍与一位已经 ac 的舍友交流后(不是那个卡常 ac 的)，得知输入格式中有科学计数法，修改了快读的代码后，成功突破 79：

提交 #3665759

#	送交者	题目	语言	提交时间	评测时间	结果	耗时	内存	评测机
3665759	10235101477	A. 相似最近邻搜索	C++17	2024-05-20 09:38:21	2024-05-20 09:38:38	Time limit exceeded: 86	1.997	7.680	Atlanda

在突破了 79 之后，我将一些循环展开，并将所有的非浮点数运算改为位运算，成功突破 86：

提交 #3665924

#	送交者	题目	语言	提交时间	评测时间	结果	耗时	内存	评测机
3665924	10235101477	A. 相似最近邻搜索	C++17	2024-05-20 12:48:25	2024-05-20 12:48:40	Time limit exceeded: 93	1.760	7.828	Atlanda

最后的一个用例的突破所花时间最久，不过我还是没卡出来，最后询问了那位卡常成功的同学后，加上火车头，便可以 ac：

提交 #3668763

#	送交者	题目	语言	提交时间	评测时间	结果	耗时	内存	评测机
3668763	10235101477	A. 相似最近邻搜索	C++17	2024-05-22 15:50:15	2024-05-22 15:50:28	Accepted: 100	1.317	19.844	Atlanda

该算法的 testcase 数据测试：

```
//精确搜索(优先队列方式)
第一次：16.330s；第二次：16.279s;第三次：16.621s;
```

速度相比于 kd 树慢了 4s，比球速快多了。
Sift 数据集：

```
第一次：7881.278s；  第二次：；5192.072s;
```

第一次是挂后台跑，第二次是挂前台跑，两次数据相差较远，这个挂前台跑和挂后台跑的区别是这次实验的意外收获，不过整体还是比较快的，如果挂前台跑，两个小时内能跑完。

6. 快速选择：

思路：
该算法是先将查询向量与数据集中所有向量的距离算出，再通过随机选 pivot 的方式，进行快排，每排完一次，就判断这个 pivot 前面有多少个向量，如果正好 k 个，那就直接返回，如果小于 k 个，就只对后面的部分进行快选，大于 k 个，就只对前面 k 个快选，直到选出 k 个为止。
时间复杂度：平均：O(n)；最坏：O(n^2);
空间复杂度：O(1);
算法评价与概述：
这个算法也是 topk 的热门解法，也不是 gpt 给的（gpt 给的算法就没一个能过的。。。），属于 KNN，我是在 b 站上先自学了快速选择的思路（[O(N) 时间解决 Top K 问题 | 小旭详解 基础算法系列 第 K 大的数 快速选择 - EP8_哔哩哔哩_bilibili]），由于学过快排，只要将快排的代码稍作修改即可，以下是 eoj 上的提交：

提交 #3678112

#	送交者	题目	语言	提交时间	评测时间	结果	耗时	内存	评测机
3678112	10235101477	A. 相似最近邻搜索	C++17	2024-05-30 14:24:26	2024-05-30 14:24:41	Time limit exceeded: 71	2.000	7.293	Atlanda

（注意：本次提交与优先队列的 ac 代码一样使用了快读快写和火车头）
理论上快速选择的速度要略优于优先队列，但是在实际测试的过程中，快速选择是要比优先队列稍慢的，下面是 testcase 和 sift 的数据集测试：
Testcase 的数据测试：

```
* //精确搜索(快速选择)
* 第一次：20.927s；第二次：22.050s；第三次：20.834s
```

平均时间在 21s 左右；
Sift 的数据测试：

```
第一次：8944.276s；第二次：5215.356s
```

第一次是挂后台跑的，第二次是挂前台跑的，两次的数据均慢于优先队列的速度。

关于快速选择，还有一个有意思的现象，当时我让 gpt 写了一个快速选择的代码，本以为 gpt 写的超不过 30 分(gpt 写的其他算法最高分为 21 分)，提交后 gpt 的达到了 50 分：

提交 #3668221

#	送交者	题目	语言	提交时间	评测时间	结果	耗时	内存	评测机
3668221	10235101477	A. 相似最近邻搜索	C++17	2024-05-21 22:16:27	2024-05-21 22:16:48	Time limit exceeded: 50	2.004	8.641	Atlanda

这个是令我意外的，可能过不久 AI 真能替代程序员。。。

7. LSH：

思路：
该算法是利用哈希函数将数据集中相邻的数据映射到尽可能近的区域（尽可能制造哈希冲突），例如，将相邻的数据映射到一个桶中，等到查询数据的时候，只要将这个查询向量通过哈希函数进行映射，找到对应的桶，并对该桶和邻近的桶中的数据进行计算，得到前 k 个返回。
时间复杂度：与设置的哈希函数和桶的个数有关；
空间复杂度：与设置的哈希函数和桶的个数有关；

算法评价与概述：
该算法是最抽象的一个，查了一个周末的资料还没完全理解，后面按照网上的一个哈希函数（利用向量的点积进行映射），自己写出了一个版本，下面是 eoj 提交：

提交 #3653026

#	送交者	题目	语言	提交时间	评测时间	结果	耗时	内存	评测机
3653026	10235101477	A. 相似最近邻搜索	C++17	2024-05-11 20:16:55	2024-05-11 20:17:23	Time limit exceeded: 14	2.005	3.059	Atlanda

只过了两个，在这之后我询问了助教关于 LSH 算法的问题，助教认为 LSH 算法召回率太低，不是这个问题的最优解，此外，我想不出其他的哈希函数，于是接下来的测试是以 gpt 的代码测试为主。
Gpt 的 eoj 提交：

提交 #3660817

#	送交者	题目	语言	提交时间	评测时间	结果	耗时	内存	评测机
3660817	10235101477	A. 相似最近邻搜索	C++17	2024-05-18 11:35:34	2024-05-18 11:35:52	Time limit exceeded: 21	2.010	31.863	Atlanda

Testcase 的数据:

```
* //LSH
* 第一次：23.057s；第二次：23.065s；第三次：19.692s
```

速度与快速选择差不多，召回率相比于 kd 树高，大部分数据的召回率为 70%~100%;

Sift 的数据:

```
* 第一次：11019.721s;第二次：8552.323s;
```

前一次是挂后台跑的，后一次是挂前台跑的，速度均慢于优先队列和快速选择，其召回率却奇低，几乎只有 0 和 1%.

8. HNSW:

思路:

将数据集构建成一个近邻图，该近邻图是一个分层的图，对于每一个插入的向量，会根据特定的算法算出该向量应插入的层数，根据我查的资料([HNSW 算法的基本原理及使用_hnws 算法-CSDN 博客](), [HNSW 算法详解-CSDN 博客]()), 越靠近顶层，点之间的距离越大，点的个数也越少，这样，对于每一个查询向量，可以先通过少量的点快速确定最近邻的大致范围，接着到下一层，在更密的点里进一步缩小范围，最终找到近似的 k 个最近邻。

时间复杂度：平均：O((logn)^2);

空间复杂度：O(n*m*logn);//m 为一层的平均邻居个数;

算法评价与概述:

该算法所花时间也较久，也是和 LSH 一样，查完还是很懵圈，尤其是对插入向量对应插入层位置的算法，没有中文资料讲得详细的(可能是我看不懂？)，在花了两个早上看资料无果的情况下，我将代码交给了 gpt，以下是 eoj 上用 gpt 代码的提交:

提交 #3673855

#	送交者	题目	语言	提交时间	评测时间	结果	耗时	内存	评测机
3673855	10235101477	A. 相似最近邻搜索	C++17	2024-05-27 15:35:44	2024-05-27 15:36:12	Time limit exceeded: 7	2.005	11.074	Atlanda

该算法只过了一个，且全部超时，我将唯一通过的数据与线性搜索的通过的数据所花的时间进行对比(二者通过的用例是一样的)，该方法只比线性搜索快了 0.2s

在随后的 testcase 数据集测试中，HNSW 测试的总时间超过了 10 分钟，便没有再测试 sift 数据集和其召回率。

一些感想:

1. 在很大的数据下挂前台跑和挂后台跑差别那么大是我没想到的，以后跑程序还是挂前台跑;
2. 晚上跑程序的时候一定要关闭 windows 更新。。。跑到半路更新了，网卡就会用不了，然后就要花时间重装系统。。。血的教训。。。