Lab1 Learned Index 报告

2025 年 3 月 9 日

1 背景介绍

键值对是一种数据存储的基本结构，由“键”和“值”两个部分构成。数据索引指在这种数据结构中通过“键”找到“值”的过程。这个过程可以被抽象为一个函数映射 [1]。

C++ 中的 std::map 底层使用了红黑树。红黑树是一种接近平衡的二叉搜索树，其中每一个节点都是红色或黑色。更具体地说，根节点是黑色、叶子节点是黑色、红色节点的子节点是黑色，并且从任意节点到其每个叶子的所有路径都包含相同数量的黑色节点。这些规则保证红黑树相对平衡，高度大概保持在对数级别，避免了二叉查找树退化为链表的情况。

Linear model 和 decision model 都使用了深度学习的理念。在 linear model 中，对“键”排序后，程序将“键”与“值”所存储的位置拟合为线性函数。在查找键对应的值时，首先用函数预测值可能在的位置，在小范围内搜寻；如果搜不到则在所有键值对中搜寻。在 decision model 中，将数据由小到大排序，划分成若干个子节点，非叶子节点以类似二叉搜索树的方式组织，叶子节点使用 linear model 训练和预测。

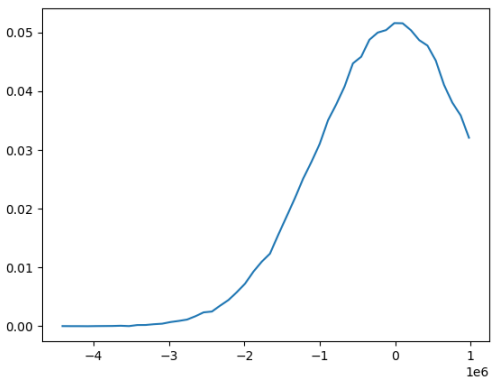
Lab 对比了这三种索引方式在不同数据分布下的读取效率。

2 测试

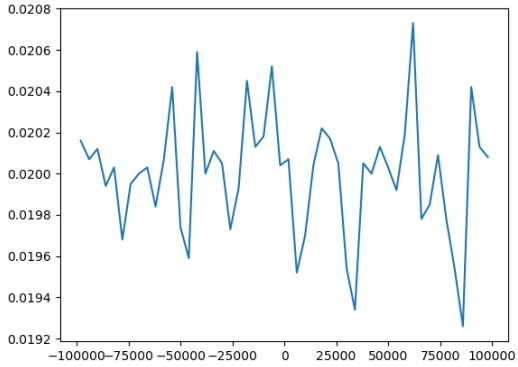
2.1 实验设置

实验运行在 Linux 环境中，使用 ctime 头文件中的 clock() 函数计算所用时间，通过读（或取）前后的时间差判断性能。

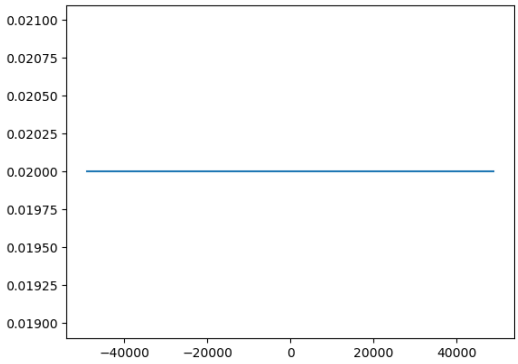
为考虑各种情况下的性能，本实验设计了三种数据分布，分为正态分布、随机分布、线性递增分布三种。图 1是这三种分布的键频率分布图。



(a) 正态分布



(b) 随机分布



(c) 线性递增分布

图 1: 四种分布的键频率

2.2 预期结果

本实验统计了这三种索引方式的读取时间。注意读入数据时，DataLoader 类从 csv 文件中读入数据，并对数据进行了排序，这部分时间不计入。更具体地说，linear model 和 decision model 的读入时间只统计训练用时。

std::map 底层使用红黑树实现，其读取时间复杂度均为 O(logn)。对于各种数据分布，红黑树都会保证树高在 O(logn) 的范围内，因此在不同数据分布下的时间复杂度将大概相同，即对于所有数据的总读取时间复杂度均为 O(nlogn)。

Linear model 会拟合线性函数，拟合的时间复杂度为 O(n)，总体时间复杂度度为 O(n)。查询单个数据时，在最坏情况下，如果查找的“键”不存在，则无论如何都会遍历整个数据结构，

复杂度 O(n)。在其它情况下，也就是“键”存在时，复杂度与数据分布有关。显然，在线性递增情况下，复杂度为 O(1)，其它情况下复杂度最好为 O(1)，最差为 O(n)。

Decision model 会构造树和拟合数据，拟合的时间复杂度与 linear model 相同。构造树时，时间复杂度为 O(logn)。也就是说，可以近似认为 decision model 的读入时间复杂度为 O(n)，但实际读入时间应长于 linear model。令每个叶结点中存储的键值对数量为 m，则最好情况下在叶子节点中读取单个数据复杂度为 O(1)，最差为 O(m)。实际读取时，还需要以 O(logn) 的时间复杂度找到所需叶子节点，读取复杂度为 O(logn) 到 O(logn + m) 之间，与数据分布、n 和 m 的值都有关。此外，易知对于正态分布等，decision model 可以更关注小部分数据的统计规律，查询性能好于 linear model。

综上，预期结果为，对于写入性能，linear model 最优，std::map 最差；对于读取性能，在大多数情况下 decision model、std::map 优于 linear model，在线性递增分布下 linear model 优于 decision model，decision model 优于 std::map。此外，decision model 和 std::map 的读取性能与 decision model 参数、数据分布相关。

2.3 实验结果与分析

令总数据数量分别为 10000 和 100000。这三种结构在不同数据分布下的写入及读取时间如表 1、表 2。

表 1: n=10000 时的读写用时

n = 10000	正态分布	随机分布	线性递增分布
std::map 写入	8.652ms	9.081ms	8.670ms
linear model 写入	0.081ms	0.087ms	0.082ms
decision model 写入	2.711ms	1.957ms	1.878ms
std::map 读取	4.432ms	4.945000ms	4.134ms
linear model 读取	996.507ms	1016.499ms	4.865ms
decision model 读取	6.701ms	7.043ms	6.291ms

表 2: n=100000 时的读写用时

n = 100000	随机分布	正态分布	线性递增分布
std::map 写入	121.066	119.036ms	133.263ms
linear model 写入	1.214ms	1.166ms	1.107ms
decision model 写入	18.857ms	20.588ms	21.105ms
std::map 读取	60.837ms	64.679ms	80.921ms
linear model 读取	98384.079ms	97308.524ms	70.691ms
decision model 读取	90.646ms	118.890ms	74.772ms

可以看出，正态分布和随机分布下，数据基本与预期结果相符：std::map 的写入性能最劣，linear model 的写入性能最优；std::map 的读取性能最优，linear model 的读取性能最劣。

linear model 在线性递增分布中的表现远远好于它在另外两种分布中的表现，这与实验预期相符。但在其它方面，线性递增分布得到的读取时间表现出较强的“随机性”，linear model 基本上好于 decision model，至于 linear model 孰优孰劣，测试结果并不稳定。这种情况可能与许多因素有关，例如缓存、CPU 占用等。

3 结论

此 Lab 通过 ctime 中的 clock 函数，在 linux 环境下统计了 std::map、linear model、decision model 三种索引方式在不同数据分布下的性能。

在 n=10000 和 100000 的情况下，从实验结论来看，linear model 和 decision model 的写入性能显著优于 std::map。在大部分情况下，linear model 的读取性能低，但在线性分布数据下表现出较好的性能。decision model 和 std::map 的读取性能都较好，且实验中 std::map 更优。

如果将同时考虑读写，则有 decision model 优于 std::map，std::map 优于 linear model。

由于缓存等原因，本实验存在一定误差。此外，实验中没有考虑键重复的情况。

4 致谢

感谢知乎等博客提供的参考；感谢 deepseek、kimi 等大模型提供的思路与帮助。

感谢提供支持的室友和朋友们。

5 其他和建议

本来在设计项目的时候，还设计了一种局部聚集分布（图 2已经做好了）

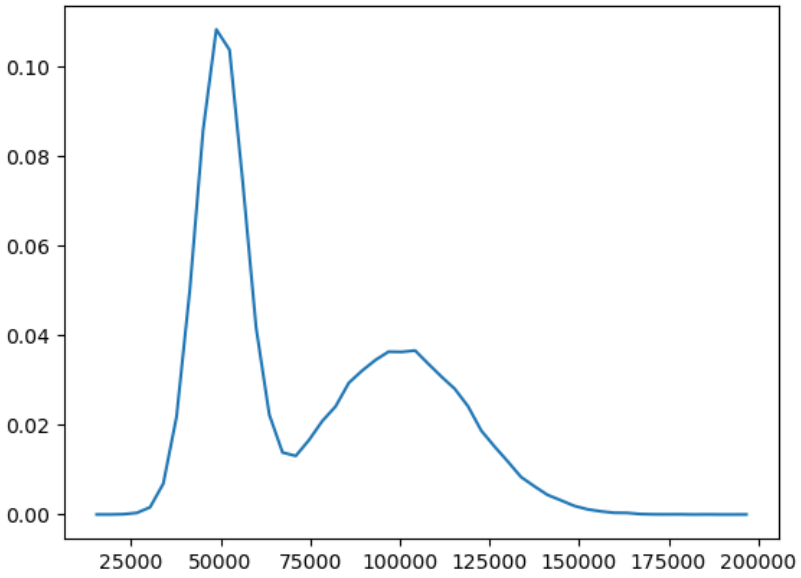


图 2: 局部聚集分布

（其实是两个正态分布拼在一起）

然后跑测试的时候才突然意识到 key 是 int 类型，这种随机数有很多重复，代码没有考虑键值重复的情况。python 试着生成符合正态分布的不重复整数（也可能是有什么方法我不知道?），感觉 100000 大小要跑一段时间。当然我也可以把 keytype 改成 double，但是 double 和 int 性能肯定不一样，那就要把其它几种分布再重跑一遍改数据（粗暴地删掉 assert 语句也同理）。这个其实也没多麻烦，不过这种分布和正态分布体现出的特点应该是差不多的，所以没有继续改，直接把这种分布删掉了。

不喜欢写报告（当然这个不是 lab 的问题，是我的问题）

还有这几天 latex 写的有点入脑……另外一门课的作业要求交 word 或者 pdf，想都没想就开了 latex。公式表格图片（还有图片）调了半天，写完之后突然意识到，为什么不手写呢:(

还有就是这个实验算是让我意识到了（计算机中）理论和实践的差距？感觉这里缓存和其它东西带来的不确定性还是比较直观的。

参考文献

[1] Tim Kraska, Alex Beutel, Ed H. Chi, Jeff Dean, and Neoklis Polyzotis. The case for learned index structures. *arXiv preprint arXiv:1712.01208*, 2017.