lab1 实验报告

王理想 522031910680

2024 年 4 月 13 日

**1** 背景介绍

基数树是⼀种多叉搜索树，能够有效地存储和检索集合（set）或映射（map）的数据。基数树通过共享相同 的前缀来节省空间，并提供快速的查找操作。由于该特点，基数树被⼴泛运⽤在IP路由、信息检索等领域 中。在Linux中基数树被⼴泛应⽤于IDR机制、page cache索引、路由查找等实现中。

但是，基数树的实现存在冗余的节点，这使得树的基本操作效率降低。具体来说，如果树中⼀个⾮根节点只有⼀个⼦节点，我们可以将该节点与其⼦节点进⾏合并，合并后的节点

所代表的值为被合并节点代表的值拼接后的结果。由于节点数减少了，在执⾏基本操作时，就可以减少在树上进⾏遍历节点的节点跳转次数。如下图左侧所示，标有相同颜⾊的节点就可以合并成⼀个节点。基于这样的观察我们可以对基数树进⾏节点压缩（node compression）。

红黑树是一种自平衡的二叉查找树，是一种高效的查找树。红黑树具有良好的效率，它可在 O(logN) 时间内完成查找、增加、删除等操作。因此，红黑树在业界应用很广泛，比如 Java 中的 TreeMap，JDK 1.8 中的 HashMap、C++ STL 中的 map 均是基于红黑树结构实现的。

下面我们将通过实验分别测试这三类树的操作效率。

**2.1** 性能测试

**2.1.1 遇到难点：**

1、YCSB测试实现的问题，不会使用ycsb测试工具，没有理解lab文档中自行实现测试的含义。

2、CompressRadixTree的删除后仅剩一个子节点的合并，开始合并到了父节点，造成子节点的子树丢失，debug用了很长时间。

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RadixTree | | | | CompressedRadixTree | | | | RBTree | | | |
| 时间/s | 平均时延 | P50 | P90 | P99 | 平均时延 | P50 | P90 | P99 | 平均时延 | P50 | P90 | P99 |
| workload1 | 8.47E-07 | 7.99E-07 | 9.49E-07 | 2.24E-06 | 7.82E-07 | 7.30E-07 | 8.90E-07 | 2.10E-06 | 1.55E-06 | 1.22E-06 | 2.30E-06 | 4.66E-06 |
| workload2 | 3.42E-07 | 3.10E-07 | 3.40E-07 | 1.28E-06 | 3.63E-07 | 3.40E-07 | 3.70E-07 | 1.31E-06 | 3.10E-07 | 2.99E-07 | 3.20E-07 | 4.61E-07 |
| workload3 | 1.12E-06 | 8.73E-07 | 1.53E-06 | 3.48E-06 | 9.42E-07 | 8.10E-07 | 1.07E-06 | 2.81E-06 | 8.84E-07 | 8.20E-07 | 1.12E-06 | 2.01E-06 |

**2.1.2 实验数据结果**

实验结果分析：

1. RBTree在workload2,3下的平均时延都小于其他两个树，但在workload1下明显弱于其他两个数，说明RBTree的插入性能远比其他两个数要差，但删除和查找性能好于其他两个树。
2. CompressedRadixTree在各个worikload下平均时延均比RadixTree小，说明前者比后者综合性能略佳。
3. 对于RBTree，在workload2下平均时延、P50、P90、P99最小，在workload1下平均时延、P50、P90、P99最高，说明在红黑树中，find都是最省时的，而insert是最耗时的。对于CompressedRadixTree和RadixTree,都在workload2下平均时延、P50、P90、P99最小，在workload3下平均时延、P50、P90、P99最高，说明在两者操作中，find都是最省时的，而remove是最耗时的。

**2.1.3** 红黑树来源：chatgpt编写

**3** 结论

这次lab让我对课上学习的这三个数据结构的性能有了更深入的认识与了解，使我受益匪浅。

**4** 致谢

感谢高级数据结构课程助教曾靖垚，许贤对lab问题的解答

**5** 其他和建议

无

参考文献

无