

1 实验参数设置

1.1 LinkedList

LinkedList中的数据为long类型的单值

1.2 SkipList

SkipList中的数据为形如 $\langle long, long \rangle$ 的键-值对，插入一个新节点时，节点向上分裂的概率为0.5

1.3 BPlusTree

BPlusTree中的数据为形如 $\langle long, long \rangle$ 的键-值对，BPlusTree的阶数为1024

1.4 LongTVList

LongTVList中的数据为形如 $\langle long, long \rangle$ 的时间戳-值对，底层链表的长度不超过128

2 顺序插入结果

将1到 n 的数据顺序插入数据结构，统计插入时间

| | 10^3 | 10^4 | 10^5 | 10^6 |
|------------|--------|--------|--------|---------|
| LinkedList | 0.2ms | 0.7ms | 1.7ms | 32.6ms |
| SkipList | 0.5ms | 1.9ms | 14.5ms | 172.6ms |
| BPlusTree | 0.9ms | 5.3ms | 15.5ms | 120.2ms |
| LongTVList | 0.4ms | 1.0ms | 3.6ms | 22.1ms |

3 随机插入结果

将1到 n 的一个排列插入数据结构，统计插入及维护数据有序的总耗时

| | 10^3 | 10^4 | 10^5 | 10^6 |
|--------------------|--------|---------|---------|------------|
| LinkedList | 12.0ms | 293.0ms | 34929ms | 21311685ms |
| SkipList | 0.5ms | 2.4ms | 32.4ms | 779.2ms |
| BPlusTree | 1.0ms | 6.4ms | 42.0ms | 825.7ms |
| LongTVList(insert) | 0.2ms | 0.6ms | 2.6ms | 18.4ms |
| LongTVList(sort) | 1.7ms | 7.8ms | 60.5ms | 747.4ms |
| LongTVList(total) | 1.9ms | 8.4ms | 63.1ms | 765.8ms |

4 插入时间分析

实验结果表明，在顺序插入时B+树表现比跳表更好，而随机插入时，跳表表现得比B+树更好，理论分析如下：

新插入一个跳表中的节点，该节点将不断以0.5的概率向上扩展，因此每一层跳表中的节点数期望是下一层的 $\frac{1}{2}$ 。在跳表中插入一个新节点，将从顶层开始，依次遍历每层节点，确定位置后转入下一层。这个过程实际上与二分查找等价，因此跳表中插入一个新节点的复杂度始终为 $O(\log_2 n)$ 。

新插入一个B+树中的节点，当插入数据为升序时，因为新数据总比之前所有数据都大，所以新数据总是插入在当前节点的最后一个子节点所在的子树内，这将触发B+树插入过程的特判机制，能够快速定位新节点的位置。因此节点度为1024的B+树在插入顺序数据时的效率能达到 $O(\log_{1024} n)$ 。对于随机数据插入，B+树需要在每一层二分查找该数据需要插入的子节点的位置，复杂度为 $O(\log_2 1024)$ ，此外，B+树的层数为 $O(\log_{1024} n)$ 。因此当插入随机数据时，B+树的插入复杂度为 $O(\log_2 1024) \times O(\log_{1024} n) = O(\log_2 n)$ ，与跳表的理论复杂度相当。但是B+树的插入过程是双层二分查找，常数复杂度高于跳表的普通二分查找，而且B+树在插入节点后具有比跳表更复杂的向上分裂过程，故当插入随机数据时，B+树的表现将不如跳表。