测试结果分析

邓朋　　17302010026

一：插入数据

B+树（本次测量使用的 B+树的 minDegree 为 5，即每个节点最少含有 4 个 key 值，最多含有 9 个 key 值）

插入初期每插入 100 个单词的运行时间：

```
class BPlusTree Insert time: 400046ns
class BPlusTree Insert time: 1111158ns
class BPlusTree Insert time: 409892ns
class BPlusTree Insert time: 389835ns
class BPlusTree Insert time: 384729ns
class BPlusTree Insert time: 1227123ns
class BPlusTree Insert time: 412809ns
class BPlusTree Insert time: 258918ns
class BPlusTree Insert time: 256001ns
class BPlusTree Insert time: 639636ns
class BPlusTree Insert time: 245061ns
class BPlusTree Insert time: 220263ns
class BPlusTree Insert time: 230473ns
class BPlusTree Insert time: 592958ns
```

插入足够数目单词后每插入 100 个单词的运行时间：

```
class BPlusTree Insert time: 535704ns
class BPlusTree Insert time: 631249ns
class BPlusTree Insert time: 229744ns
class BPlusTree Insert time: 227921ns
class BPlusTree Insert time: 209687ns
class BPlusTree Insert time: 420102ns
class BPlusTree Insert time: 223909ns
class BPlusTree Insert time: 215522ns
class BPlusTree Insert time: 204216ns
class BPlusTree Insert time: 437972ns
class BPlusTree Insert time: 248342ns
class BPlusTree Insert time: 211145ns
class BPlusTree Insert time: 219168ns
```

通过测量数据，可以观察到在插入初期，B+树的插入时间更长并且时间波动较大，在插入后期，B+树的插入时间趋于稳定且相对较短。主要原因为在 B+树的构建初期，由于节点数目较少，会进行相对较多的拆分节点操作；在插入较多数据后，B+树的结构会趋于稳定，进而每次插入的时间也会相对稳定；

红黑树：

插入初期每插入 100 个单词的运行时间：

```
class RedBlackTree Insert time: 105391ns
class RedBlackTree Insert time: 308513ns
class RedBlackTree Insert time: 170666ns
class RedBlackTree Insert time: 113049ns
class RedBlackTree Insert time: 113778ns
class RedBlackTree Insert time: 334404ns
class RedBlackTree Insert time: 160821ns
class RedBlackTree Insert time: 93721ns
class RedBlackTree Insert time: 92262ns
class RedBlackTree Insert time: 234120ns
class RedBlackTree Insert time: 99920ns
class RedBlackTree Insert time: 92262ns
class RedBlackTree Insert time: 96638ns
```

插入足够数目单词后每插入 100 个单词的运行时间：

```
class RedBlackTree Insert time: 138940ns
class RedBlackTree Insert time: 111225ns
class RedBlackTree Insert time: 261470ns
class RedBlackTree Insert time: 109402ns
class RedBlackTree Insert time: 105026ns
class RedBlackTree Insert time: 122165ns
class RedBlackTree Insert time: 256000ns
class RedBlackTree Insert time: 115966ns
class RedBlackTree Insert time: 111954ns
class RedBlackTree Insert time: 115601ns
class RedBlackTree Insert time: 242507ns
class RedBlackTree Insert time: 119613ns
class RedBlackTree Insert time: 113778ns
class RedBlackTree Insert time: 114143ns
```

对于红黑树来说，插入初期会有小范围的时间较长区域，然后插入时间会迅速下降，并逐渐趋于稳定，主要原因为红黑树在构建初期会出现较多的颜色冲突问题，进行调整会占用相对较多时间，插入较多数据后，红黑树会很快趋向平衡，并且红黑树的平衡性保持的相对较好，这也从侧面证实了红黑树是一种平衡二叉树。

红黑树和 B+树的比较：从数据可以明显看出，红黑树的插入操作花费时间大约为 B+树插入操作的二分之一，主要原因为由于 B+树节点包含较多的 key 值，需要进行较多次数的比较操作，并且由于 B+树的 value 仅存储在叶节点中，故 B+树需要更多的逻辑判断。

二：删除数据

B+树：

```
class BPlusTree Delete time: 524764ns
class BPlusTree Delete time: 873391ns
class BPlusTree Delete time: 444900ns
class BPlusTree Delete time: 410986ns
class BPlusTree Delete time: 463135ns
class BPlusTree Delete time: 507989ns
class BPlusTree Delete time: 327841ns
class BPlusTree Delete time: 267669ns
class BPlusTree Delete time: 343157ns
class BPlusTree Delete time: 233391ns
```

红黑树：

```
class RedBlackTree Delete time: 298667ns
class RedBlackTree Delete time: 269129ns
class RedBlackTree Delete time: 274963ns
class RedBlackTree Delete time: 233026ns
class RedBlackTree Delete time: 200935ns
class RedBlackTree Delete time: 150974ns
class RedBlackTree Delete time: 143316ns
class RedBlackTree Delete time: 140035ns
class RedBlackTree Delete time: 186348ns
class RedBlackTree Delete time: 142222ns
```

对比分析：对于每种树来说，随着所包含的 key 值的减少，删除操作的花费也逐渐减小。对比下，同等节点数目下，B+树的删除操作花费大约是红黑树的两倍，由于每种树在删除之间都要进行查询操作，因此可以推断出 B+树的查询操作花费也大约是红黑树的两倍。另外，B+树在删除时需要同时删除内部节点的 key 值和叶节点的 key 值与 value 值，因此 B+树相当于每次删除都需要做两次查询操作，而红黑树只需要一次就好。

三：增加数据

B+树：

```
class BPlusTree Insert time: 294291ns
class BPlusTree Insert time: 156810ns
class BPlusTree Insert time: 109766ns
class BPlusTree Insert time: 180513ns
class BPlusTree Insert time: 102109ns
class BPlusTree Insert time: 102838ns
class BPlusTree Insert time: 118154ns
class BPlusTree Insert time: 152068ns
class BPlusTree Insert time: 148422ns
class BPlusTree Insert time: 157539ns
```

红黑树：

```
class RedBlackTree Insert time: 137482ns
class RedBlackTree Insert time: 106484ns
class RedBlackTree Insert time: 126906ns
class RedBlackTree Insert time: 161550ns
class RedBlackTree Insert time: 106484ns
class RedBlackTree Insert time: 86063ns
class RedBlackTree Insert time: 86063ns
class RedBlackTree Insert time: 128365ns
class RedBlackTree Insert time: 84969ns
class RedBlackTree Insert time: 80958ns
```

同之前插入数据的情况相同，相比之下，红黑树的插入操作更快。

四：单个单词查询

B+树随机抽取 10 个单词的查询时间情况，平均每次查询大约花费 10000ns

```
BPlusTree search time: 14952ns
BPlusTree search time: 14223ns
BPlusTree search time: 6564ns
BPlusTree search time: 12034ns
BPlusTree search time: 7658ns
BPlusTree search time: 10210ns
BPlusTree search time: 9117ns
BPlusTree search time: 9481ns
BPlusTree search time: 8022ns
BPlusTree search time: 10576ns
```

红黑树随机抽取 10 个单词的查询时间情况，平均每次查询大约花费 5000ns

```
RedBlackTree search time: 8752ns
RedBlackTree search time: 4376ns
RedBlackTree search time: 5470ns
RedBlackTree search time: 7294ns
RedBlackTree search time: 5106ns
RedBlackTree search time: 4011ns
RedBlackTree search time: 6928ns
RedBlackTree search time: 4740ns
RedBlackTree search time: 12399ns
RedBlackTree search time: 5834ns
```

同在删除操作时的分析相同，平均情况下，红黑树的查询效率是 B+树的两倍。主要原因为 B+树的 value 仅保存在叶节点当中，故 B+树的每次查询都需要查找到叶节点为止，而红黑树的 key 值和 value 值都保存在内部节点当中，每次查询的花费并不相同。并且 B+树由于节点含有较多的 key 值，需要更多次数的比较操作。

五：范围查询

B+树与红黑树在相同查询范围时的时间对比：

```
BPlusTree range search time: 1200867ns
RedBlackTree range search time: 1067762ns
BPlusTree range search time: 727522ns
RedBlackTree range search time: 617755ns
BPlusTree range search time: 475168ns
RedBlackTree range search time: 360661ns
```

和之前的几种操作相比，B+树在范围查询方面表现得更好一点，主要原因为 B+的叶节点之间是直接相连的，因此在查找到第一个单词后可以直接找到叶节点的右节点继续查找，相对于单个单词查询时花费为红黑树的两倍来讲，B+树在范围查询上面表现得更好一点。

总结：

红黑树是一种平衡二叉树，对于常用的插入、查询、删除操作都有较好的表现；B+树的主要作用为在数据量较大而导致内存不能完全存储时，可以有效的减少内存和磁盘的交换操作。而在本次的实验中，并没有涉及到内存和磁盘的交换，因而 B+树的优势并没有完全表现出来。另外，由于 B+树具有节点中储存的 key 值数目的灵活性，因此不同的 B+树可能会具有不同的表现。