1. 编译和运行环境

编译环境：Windows 7，DevC++ TDM-GCC 4.9.2 64-bit Release

编译参数：-static-libgcc -std=c11

运行环境：64位Windows，i5-6200U，DDR3L 1600MHz，5400转机械硬盘

1. 算法和思路

此实验主要涉及文件IO，查找和排序。为保证运行速度，首先应确保文件IO不能严重阻塞后续操作，接下来应使得查找和排序尽量快速。于是考虑过以下策略：

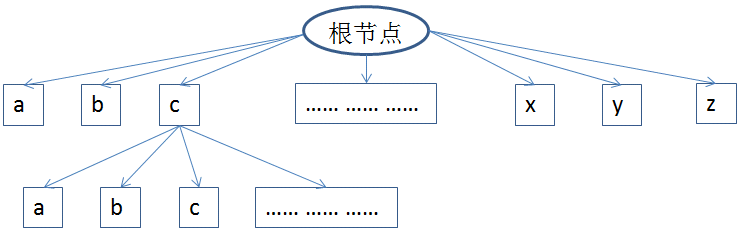
* 1. 内存文件映射

使用标准的fopen，fgetc等函数读取大文件时，会在IO上耗费过多时间。这类函数常常先把文件读入缓冲区，再从缓冲区复制内容到内存。事实上我们可以通过内存文件映射，为磁盘块中的文件每一地址分配对应的内存地址。此时文件并没有被载入内存，即被映射的内存中存放的可以是任意内容。当我们去读取对应的内存时，操作系统将为我们调用硬盘中的文件内容。读取的内容并不是被复制到内存中的映射空间，而是可直接在寄存器中被程序使用。类似的，当我们要求向映射空间写入内容时，操作系统将（自行选择适当时机）直接写入硬盘对应位置。建立映射所花的时间与文件本身的体积相关不大，因此这一IO方式可有效提升大文件的IO效率。但对于小文件，建立映射的过程消耗的时间可能显得过多。

不同操作系统下，实现内存文件映射的方式是不同的。本次实验只写了Windows下映射的代码，因此程序不能直接在Linux系统编译。

* 1. 字典树

字典树用于查找一个单词是否已经出现过，以及对于出现过的单词查找修改其出现次数。它的数据结构如下图：



字典树每个节点包括27个指针，其中26个字母指针用于指向其他字典树节点。另外1个指示指针用于指示这个单词是否出现过。字典树最初只有一个根节点。当我们开始查找一个单词（例如”cat”）时，我们首先拿出字母c（即第3个字母），查找根节点的第3个字母指针是否为NULL。如果是NULL，则将在根节点下新建节点c。接下来再取出cat中第二个字母a，查找c下是否存在a。最后程序将停在a下的节点t，并查询节点t的指示指针。如果指示指针为NULL说明单词cat之前没有出现过。于是我们为单词cat新开一块内存空间记录其出现次数，并挂靠在节点t的指示指针下。利用字典树，我们可以将查找单词的时间控制在O(单词长度)内。此外，字典树便于按字典顺序输出单词。

* 1. 散列（哈希）

散列方法是根据字符串本身内容，以某种算法直接求出一个内存位置，将字符串放入。同一字符串散列的结果是唯一的。当两个不同字符串的散列结果相同时，我们可以用链表将两个字符串挂靠在同一内存位置。散列方法的查找时间大概率为O(1)，但从字符串内容计算其内存位置的算法应当简单快速，且要降低不同单词出现在同一内存位置的概率。

与字典树相比，散列法通过额外消耗大量内存来节约少量时间。若散列求取内存位置的算法过于复杂，则其消耗的时间很容易超出字典树查找时间，从而失去速度优势。散列的可扩展性远不及字典树，仅适用于高度定制化的系统。实际应用中，或许字典树在大多数情况是更好的选择。

* 1. 堆排序和桶排序

堆排序消耗时间为O(n logn)，其中n是不同单词的个数。堆排序不会消耗过多额外内存，且具有稳定性。桶排序消耗时间O(m)，其中m是单词出现次数的取值范围（实际测试发现，某些单词出现的次数远超不同单词的个数n）。同时，桶排序额外占用O(m)的内存空间，具有稳定性。比较之下，认为堆排序节约内存的特性更适合字典树查找完成后的排序，而桶排序的快速性更适合散列后的排序。这是由字典树和散列方法的本身特性决定的。字典树是在相对有限的内存下取得不太慢的速度，而散列是用海量内存换取极限速度。排序算法应延续查找算法的设计初衷，从而在实际应用中取得更好效果。

1. 耗时

最终写了三套程序，分别是字典树+堆排序(wordCount.c)、散列+堆排序(wordCountHashHeap.c)、散列+桶排序(wordCountHashBucket.c)。每套程序都使用内存文件映射来读文件，fprintf来写文件。

不使用O3优化的情况下，字典树+堆排序通常耗时0.16到0.19秒，散列+堆排序耗时0.14到0.17秒，散列+桶排序耗时0.12秒到0.17秒。

使用O3优化后，字典树+堆排序耗时0.11到0.16秒，散列+堆排序耗时0.11到0.12秒，散列+桶排序耗时0.093到0.12秒。

试验表明每一种方法的查找时间都远多于排序和输出时间。这导致本实验中桶排序相对堆排序没有明显速度优势。此外，由字符串内容求内存地址的散列算法应追求高度简洁（即使这会增加冲突概率）。稍复杂的算法就会明显增加耗时。

1. 进一步提升性能的可能性

假设输入文件体积很大时，整套程序将阻塞于硬盘的读写时间。要对极大的输入文件进一步提升性能，应当使读硬盘和查找操作并行运行。此外堆排序和桶排序也有并行做法来提高速度。对于多个不同文件输入，可采用异步方法来读写硬盘、查找排序。