# 字典树总结

**在实际生活中，图书馆、书店都会遇到一种麻烦，就是不知道如何储存所有的书名，因为藏书实在是太多了。这种情形非常常见。**

**由此单词查找树便出现了：Trie树，是一种树形结构，是一种哈希树的变种，它通过储存大部分字符串的前缀来达到不仅是存储空间而且是查找效率的优化。**

**字典树的操作大致分为三种：插入字符串、查找字符串、删除字符串。**

**首先我们要构造一棵树。根节点编号设置为1。这个结点没有实际意义，只作为查找的起点。**

**注意：虽然有些结点存在，但有可能并不是由这个结点所对应的字符串所生成的。打个比方：我添加了“memset”字符串，m->e结点已经生成，但我查找“me”时，这个字符串是否存在呢？**

**因此，我们还需要开一个exist数组，用于记录一个字符串的结尾，遍历到这个结点，而且此结点exist，才能说明这个字符串存在。**

1. **插入字符串**

**基本思路是一个个字符进行比对，如果有相应的结点，就直接进入，直到找不到下一个结点时，再在这个结点上生枝。这个操作会同时用到查找的相关操作，所以可以和查找操作结合起来。注意：字符串插入成功后要把结尾的结点exist标为1。**

1. **查找字符串**

**鉴于插入操作也要用到这步，我们可以把函数的返回值设置为：如果找到字符串就返回0，否则返回最后一次匹配到的结点位置。**

1. **删除字符串**

**将本结点删除，并不代表它的树枝也要被剪掉，所以直接查找字符串末尾的结点位置，exist标记为0即可。**

**另外：生成的结点可以被压缩，比如某些经常出现的词缀可以被压缩成一个结点，这样的编程复杂度的确不小，但是可以节省不少的空间。**