**5.2 单词查找树**

本节所讨论的算法的性能：

查找命中所需的时间与被查找的键的长度成正比；

查找未命中只需检查若干个字符。

以字符串为键的符号表的API

创建一个符号表：StringST()

向表中插入键值对：put(String key, Value val)

键key所对应的值：get(String key)

删除键key：delete(String key)

表中是否保存着key的值：contains(String key)

S的前缀中最长的键：String longestPrefixOf(String s)

所有以s为前缀的键：Iterable<String> keysWithPrefix(String s)

所有和s匹配的键：Iterable<String> keysThatMatch(String s)

键值对的数量：int size()

符号表中的所有键：Iterable<String> keys()

**5.2.1 单词查找树**

和各种查找树一样，单词查找树也是由链接的结点所组成的数据结构。

每个结点都含有R条链接，其中R为字母表的大小。

每条链接都对应着一个字符，我们将每个键所关联的值保存在该键的最后一个字母所对应的的结点中。

键的尾字符所对应的的节点中的值为空，或者查找结束于一条空链接，这是一次未命中的查找。

在单词查找树中，键是由从根节点到含有非空值的结点的路径所隐式表示的。

将基于含有R个字符的字母表的单词查找树称为R向单词查找树。

编程：基于单词查找树的符号表

编程：实现keys()，keysWithPrefix(String s)和collect接口

编程：实现通配符匹配的接口keysThatMatch(String pat)

编程：实现最长前缀的接口longestPrefixOf(String s)

编程：实现删除操作的接口delete(String key)

**5.2.2 单词查找树的性质**

单词查找树的链表结构和键的插入或删除顺序无关：对于任意给定的一组键，其单词查找树是唯一的。

在单词查找树中查找一个键或是插入一个键时，访问数组的次数最多为键的长度加1。

字母表的大小为R，在一棵由N个随机键构造的单词查找树中，未命中查找平均所需检查的结点数量为logRN。

一棵单词查找树中的链接总数在RN到RNw之间，其中w为键的平均长度。

**5.2.3 三向单词查找树**

为了避免R向单词查找树过度的空间消耗，我们来学习另一种数据表示方法：三向单词查找树。

在三向单词查找树中，每个结点都含有一个字符，三条链接和一个值。

这三条链接分别对应着当前字母小于，等于和大于结点字母的所有键。

编程：基于三向单词查找树的符号表

**5.2.4 三向单词查找树的性质**

由N个 平均长度为w的字符串构造的三项单词查找树中的链接总数在3N到3Nw之间。

在一棵由N个随机字符串构造的三向单词查找树中，查找未命中平均需要比较字符~lnN次。除~lnN次外，一次插入或命中查找会比较一次被查找的键中的每个字符。

由N个随机字符串构造的根节点进行了Rt向分支且不含有外部单向分支的三向单词查找树中，一次插入或查找操作平均需要进行约lnN-tlnR次字符比较。

**5.2.5 应该使用字符串符号表的哪种实现**

参考文档：各种字符串查找算法的性能特点