常使用一些自顶向下的红黑树，这种树我们将在 12.2 节讨论。

实现 TreeSet 和 TreeMap 的一个重要问题是提供对迭代器类的支持。当然，在内部，迭代器保留到迭代中“当前”节点的一个链接。困难部分是到下一个节点高效的推进。存在几种可能的解决方案，其中的一些方案叙述如下：

1. 在构造迭代器时，让每个迭代器把包含诸 TreeSet 项的数组作为该迭代器的数据存储。这有不足，因为我们还可以使用 toArray，并不需要迭代器。

2. 让迭代器保留存储通向当前节点的路径上的节点的一个栈。根据该信息，可以推出迭代器中的下一个节点，它或者是包含最小项的当前节点右子树上的节点，或者包含其左子树当前节点的最近的祖先。这使得迭代器多少有些大，并导致迭代器的代码臃肿。

3. 让查找树中的每个节点除存储子节点外还要存储它的父节点。此时迭代器不至于那么大，但是在每个节点上需要额外的内存，并且迭代器的代码仍然臃肿。

4. 让每个节点保留两个附加的链：一个通向下一个更小的节点，另一个通向下一个更大的节点。这要占用空间，不过迭代器做起来非常简单，并且保留这些链也很容易。

5. 只对那些具有 null 左链或 null 右链的节点保留附加的链。通过使用附加的布尔变量使得这些例程判断是一个左链正在被用作标准的二叉树左链还是一个通向下一个更小节点的链，类似地，对右链也有类似的判断(见练习 4.50)。这种做法叫做线索树(threaded tree)，用于许多平衡二叉查找树的实现中。

4.8.4 使用多个映射的例

许多单词都和另外一些单词相似。例如，通过改变第 1 个字母，单词 wine 可以变成 dine、fine、line、mine、nine、pine 或 vine。通过改变第 3 个字母，wine 可以变成 wide、wife、wipe 或 wire。通过改变第 4 个字母 wine 可以变成 wind、wing、wink 或 wins。这样我们就得到 15 个不同的单词，它们仅仅通过改变 wine 中的一个字母而得到。实际上，存在 20 多个不同的单词，其中有些单词更生僻。我们想要编写一个程序以找出通过单个字母的替换可以变成至少 15 个其他单词的单词。假设我们有一个词典，由大约 89 000 个不同长度的不同单词组成。大部分单词在 6～11 个字母之间。其中 6 字母单词有 8 205 个，7 字母单词有 11 989 个，8 字母单词 13 672 个，9 字母单词 13 014 个，10 字母单词 11 297 个，11 字母单词 8 617 个(实际上，最可变化的单词是 3 字母、4 字母和 5 字母单词，不过，更长的单词检查起来更耗费时间)。

最直接了当的策略是使用一个 Map 对象，其中的关键字是单词，而关键字的值是用 1 字母替换能够从关键字变换得到的一列单词。图 4-64 中的例程显示最后得到的(我们必须写出这部分的代码)Map 如何能够用来打印所要求的答案。该程序得到项的集合并使用增强的 for 循环遍历该项集合并观察这些由一个单词和一列单词组成的序偶。

主要的问题是如何从包含 89 000 个单词的数组构造 Map 对象。图 4-65 中的例程是测试除一个字母替换外两个字母是否相等的简单函数。我们可以使用该例程以提供最简单的 Map 构造算法，它是所有单词序偶的蛮力测试。这个算法如图 4-66 所示。

为了遍历单词的集合，可以使用一个迭代器，但是，因为我们正在通过一个嵌套(即多次)循环遍历该集合，因此使用 toArray 将该集合转储到一个数组(第 9 行和第 11 行)。尤其是，这避免了重复调用以使从 Object 向 String 转化，如果使用泛型那么它将发生在幕后。而我们这里则是直接给 String[]对象添加下标来使用。

如果我们发现一对单词只有一个字母不同，那么可以在 16 行和 17 行更新该 Map 对象。在私有的 update 方法中，我们在第 26 行能够看到，是否已经存在一列与关键字相关的单词，如果前面已经见过 key，因为 lst 不是 null，那么它就在这个 Map 对象中，而我们只需将该新单词添加