```
for each group g, containing words of length len
    for each position p (ranging from 0 to len-1)
    {
        Make an empty Map<String,List<String> > repsToWords
        for each word w
        {
            Obtain w's representative by removing position p
            Update repsToWords
        }
        Use cliques in repsToWords to update adjWords map
    }
```

```
// Computes a map in which the keys are words and values are Lists of words
// that differ in only one character from the corresponding key.
// Uses a quadratic algorithm (with appropriate Map), but speeds things by
// maintaining an additional map that groups words by their length.
public static Map<String,List<String>>
computeAdjacentWords( List<String> theWords )
{
    Map<String,List<String>> adjWords = new TreeMap<String,List<String>>( );
    Map<Integer,List<String>> wordsByLength =
                                   new TreeMap<Integer,List<String>>( );

      // Group the words by their length
    for( String w : theWords )
        update( wordsByLength, w.length( ), w );

      // Work on each group separately
    for( List<String> groupsWords : wordsByLength.values( ) )
    {
        String [ ] words = new String[ groupsWords.size( ) ];

        grqupsWords.toArray( words );
        for( int i = 0; i < words.length; i++ )
            for( int j = i + 1; j < words.length; j++ )
                if( oneCharOff( words[ i ], words[ j ] ) )
                {
                    update( adjWords, words[ i ], words[ j ] );
                    update( adjWords, words[ j ], words[ i ] );
                }
    }

    return adjWords;
}
```

图 4-67 计算一个映射的函数，该映射以单词作为关键字并且以只有一个字母不同的一列单词作为关键字的值。将单词按照长度分组。该算法对 89 000 个单词的词典运行 51 秒

图 4-68 包含该算法的一种实现，其运行时间改进到 4 秒。虽然这些附加的 Map 使得算法更快，而且句子结构也相对清晰，但是程序没有利用到该 Map 的关键字保持有序排列的事实，注意到这一点很有趣。