到这个 Map 的 List 中去，这件工作是通过调用第 33 行的 add 完成的。如果以前从未见过 key，那么第 29 行和 30 行则将其放到该 Map 对象中，List 大小为 0，因此 add 将该 List 大小更新为 1。总之，这是标准的保留一个 Map 的惯用做法，其中的值是一个集合。

```
1  public static void printHighChangeables( Map<String,List<String>> adjWords,
2                                           int minWords )
3  {
4      for( Map.Entry<String,List<String>> entry : adjWords.entrySet( ) )
5      {
6          List<String> words = entry.getValue( );
7
8          if( words.size( ) >= minWords )
9          {
10             System.out.print( entry.getKey( ) + " (" );
11             System.out.print( words.size( ) + "):" );
12             for( String w : words )
13                 System.out.print( " " + w );
14             System.out.println( );
15         }
16     }
17 }
```

图 4-64　给出包含一些单词作为关键字和只在一个字母上不同的一列单词作为关键字的值，输出那些具有 minWords 个或更多个通过 1 字母替换得到的单词的单词

```
1  // Returns true if word1 and word2 are the same length
2  // and differ in only one character.
3  private static boolean oneCharOff( String word1, String word2 )
4  {
5      if( word1.length( ) != word2.length( ) )
6          return false;
7
8      int diffs = 0;
9
10     for( int i = 0; i < word1.length( ); i++ )
11         if( word1.charAt( i ) != word2.charAt( i ) )
12             if( ++diffs > 1 )
13                 return false;
14
15     return diffs == 1;
16 }
```

图 4-65　检测两个单词是否只在一个字母上不同的例程

该算法的问题在于速度慢，在我们的计算机上花费 96 秒的时间。一个明显的改进是避免比较不同长度的单词。我们可以把单词按照长度分组，然后对各个分组运行刚才提供的程序。

为此，可以使用第 2 个映射！此时的关键字是个整数，代表单词的长，而值则是该长度的所有单词的集合。我们可以使用一个 List 存储每个集合，然后应用相同的做法。程序如图 4-67 所示。第 9 行是第 2 个 Map 的声明，第 13 行和第 14 行将分组置入该 Map，然后用一个附加的循环对每组单词迭代。与第 1 个算法比较，第 2 个算法只是在边际上编程困难，其运行时间为 51 秒，大约快了一倍。