```
                if ((e[elen - 1] == '.' || s[slen - 1] == e[elen - 1])) {
                    dp[slen - 1][elen - 1] = true;
                }
            }
        }
        return dp;
    }
```

字典树（前缀树）的实现

【题目】

字典树又称为前缀树或 Trie 树，是处理字符串常见的数据结构。假设组成所有单词的字符仅是“a”～“z”，请实现字典树结构，并包含以下四个主要功能。

- void insert(String word)：添加 word，可重复添加。
- void delete(String word)：删除 word，如果 word 添加过多次，仅删除一个。
- boolean search(String word)：查询 word 是否在字典树中。
- int prefixNumber(String pre)：返回以字符串 pre 为前缀的单词数量。

【难度】

尉 ★★☆☆

【解答】

字典树的介绍。字典树是一种树形结构，优点是利用字符串的公共前缀来节约存储空间，比如加入"abc"、"abcd"、"abd"、"b"、"bcd"、"efg"、"hik"之后，字典树如图 5-1 所示。

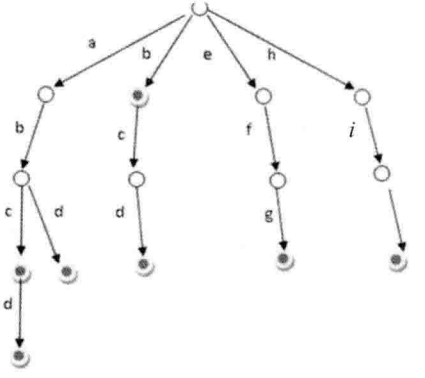

图 5-1

字典树的基本性质如下：

- 根节点没有字符路径。除根节点外，每一个节点都被一个字符路径找到。
- 从根节点到某一节点，将路径上经过的字符连接起来，为扫过的对应字符串。
- 每个节点向下所有的字符路径上的字符都不同。

在字典树上搜索添加过的单词的步骤为：

1．从根节点开始搜索。

2．取得要查找单词的第一个字母，并根据该字母选择对应的字符路径向下继续搜索。

3．字符路径指向的第二层节点上，根据第二个字母选择对应的字符路径向下继续搜索。

4．一直向下搜索，如果单词搜索完后，找到的最后一个节点是一个终止节点，比如图 5-1 中的实心节点，说明字典树中含有这个单词，如果找到的最后一个节点不是一个终止节点，说明单词不是字典树中添加过的单词。如果单词没搜索完，但是已经没有后续的节点了，也说明单词不是字典树中添加过的单词。

在字典树上添加一个单词的步骤同理，不再详述。下面介绍有关字典树节点的类型。参见如下代码中的 TrieNode 类。

```
public class TrieNode {
        public int path;
        public int end;
        public TrieNode[] map;

        public TrieNode() {
                path = 0;
                end = 0;
                map = new TrieNode[26];
        }
}
```

TrieNode 类中，path 表示有多少个单词共用这个节点，end 表示有多少个单词以这个节点结尾，map 是一个哈希表结构，key 代表该节点的一条字符路径，value 表示字符路径指向的节点，根据题目的说明，map 为长度为 26 的数组，在字符种类较多的情况下，可以选择用真实的哈希表结构实现 map。介绍完 TrieNode 后，下面详细介绍本题的 Trie 树类如何实现。

- void insert(String word)：假设单词 word 的长度为 N。从左到右遍历 word 中的每个字符，并依次从头节点开始根据每一个 word[i]，找到下一个节点。如果找的过程中节点不存在，就建立新节点，记为 a，并令 a.path=1。如果节点存在，记为 b，令 b.path++。通过最后一个字符(word[N-1])找到最后一个节点时记为 e，令 e.path++，

e.end++。

- boolean search(String word)：从左到右遍历 word 中的每个字符，并依次从头节点开始根据每一个 word[i]，找到下一个节点。如果找的过程中节点不存在，说明这个单词的整个部分没有添加进 Trie 树，否则不可能找的过程中节点不存在，直接返回 false。如果能通过 word[N-1]找到最后一个节点，记为 e，如果 e.end!=0，说明有单词通过 word[N-1]的字符路径，并以节点 e 结尾，返回 true，如果 e.end==0，返回 false。
- void delete(String word)：先调用 search(word)，看 word 在不在 Trie 树中，若在，则执行后面的过程，若不在，则直接返回。从左到右遍历 word 中的每个字符，并依次从头节点开始根据每一个 word[i]找到下一个的节点。在找的过程中，把扫过每一个节点的 path 值减 1。如果发现下一个节点的 path 值减完之后已经为 0，直接从当前节点的 map 中删除后续的所有路径，返回即可。如果扫到最后一个节点，记为 e，令 e.path--，e.end--。
- int prefixNumber(String pre)：和查找操作同理，根据 pre 不断找到节点，假设最后的节点记为 e，返回 e.path 的值即可。

全部实现过程请参看如下代码中的 Trie 类。

```
public class Trie {
        private TrieNode root;

        public Trie() {
                root = new TrieNode();
        }

        public void insert(String word) {
                if (word == null) {
                        return;
                }
                char[] chs = word.toCharArray();
                TrieNode node = root;
                int index = 0;
                for (int i = 0; i < chs.length; i++) {
                        index = chs[i] - 'a';
                        if (node.map[index] == null) {
                                node.map[index] = new TrieNode();
                        }
                        node = node.map[index];
                        node.path++;
                }
                node.end++;
        }
```

```
    public void delete(String word) {
        if (search(word)) {
            char[] chs = word.toCharArray();
            TrieNode node = root;
            int index = 0;
            for (int i = 0; i < chs.length; i++) {
                index = chs[i] - 'a';
                if (node.map[index].path-- == 1) {
                    node.map[index] = null;
                    return;
                }
                node = node.map[index];
            }
            node.end--;
        }
    }

    public boolean search(String word) {
        if (word == null) {
            return false;
        }
        char[] chs = word.toCharArray();
        TrieNode node = root;
        int index = 0;
        for (int i = 0; i < chs.length; i++) {
            index = chs[i] - 'a';
            if (node.map[index] == null) {
                return false;
            }
            node = node.map[index];
        }
        return node.end != 0;
    }

    public int prefixNumber(String pre) {
        if (pre == null) {
            return 0;
        }
        char[] chs = pre.toCharArray();
        TrieNode node = root;
        int index = 0;
        for (int i = 0; i < chs.length; i++) {
            index = chs[i] - 'a';
            if (node.map[index] == null) {
                return 0;
            }
            node = node.map[index];
        }
        return node.path;
    }
}
```