Tries 字典树

Use Condition:

- 需要查询包含某个前缀的单词 / 字符串 是否存在
- 字符矩阵中找单词的问题

复杂度

- 时间复杂度 O(L) CRUD
- 空间复杂度 O(N * L) N 是单词数，L 是单词长度

```
class TrieNode {

  public Map<Character, TrieNode> children;

  public boolean isWord;

  public String word;

  public TrieNode() {

    // 子节点中存有当前孩子和是否是个单词
    sons = new HashMap<Character, TrieNode> ();

    isWord = false;

    word = null;

  }

}

public class Trie {

    private TrieNode root;

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

```
  public TrieNode getRoot() {
    return root;
  }

  public void insert(String word) {

    TrieNode node = root;

    for (int i = 0; i < word.length(); i++){

      char letter = word.charAt(i);

      if(!node.sons.containsKey(letter)){
        node.sons.put(letter, new TriNode());
      }
      node = node.sons.get(letter);

    }

    node.isWord = true;
    node.word = word;

  }

  public boolean hasWord(String word){

    int L = word.length();
    TrieNode node = root;

    for (int i = 0; i < L; i++){

      char letter = word.charAt(i);

      if(!node.sons.containsKey(letter)){
        return false;
      }

      node = node.sons.get(letter);

    }

    return node.isWord;
  }

  //判断prefix是否存在字典树中
  public boolean hasPrefix(String prefix){

    int L = prefix.length();
    TrieNode node = root;
    for (int i = 0; i < L; i++){

      char letter = prefix.charAt(i);

      if(!node.sons.containsKey(letter)){
```

```
          return false;
        }

        node = node.son.get(letter);

      }

      return true;
    }

  }
```