字典树

又称单词查找树，Trie树，是一种树形结构，是一种哈希树的变种。典型应用是用于统计，排序和保存大量的字符串（但不仅限于字符串），所以经常被搜索引擎系统用于文本词频统计。它的优点是：利用字符串的公共前缀来减少查询时间，最大限度地减少无谓的字符串比较，查询效率比哈希树高。

性质

根节点不包含字符，除根节点外每一个节点都只包含一个字符；

从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串；

每个节点的所有子节点包含的字符都不相同。

搜索字典项目的方法为：

(1) 从根结点开始一次搜索；

(2) 取得要查找关键词的第一个字母，并根据该字母选择对应的子树并转到该子树继续进行检索；

(3) 在相应的子树上，取得要查找关键词的第二个字母,并进一步选择对应的子树进行检索。

(4) 迭代过程……

(5) 在某个结点处，关键词的所有字母已被取出，则读取附在该结点上的信息，即完成查找。

应用场景：

（1）字符串检索

事先将已知的一些字符串（字典）的有关信息保存到trie树里，查找另外一些未知字符串是否出现过或者出现频率。

举例：

1，给出N 个单词组成的熟词表，以及一篇全用小写英文书写的文章，请你按最早出现的顺序写出所有不在熟词表中的生词。

2，给出一个词典，其中的单词为不良单词。单词均为小写字母。再给出一段文本，文本的每一行也由小写字母构成。判断文本中是否含有任何不良单词。例如，若rob是不良单词，那么文本problem含有不良单词。

3，1000万字符串，其中有些是重复的，需要把重复的全部去掉，保留没有重复的字符串。

（2）字符串搜索的前缀匹配

trie树常用于搜索提示。如当输入一个网址，可以自动搜索出可能的选择。当没有完全匹配的搜索结果，可以返回前缀最相似的可能。

Trie树检索的时间复杂度可以做到n，n是要检索单词的长度，

如果使用暴力检索，需要指数级O(n2)的时间复杂度。

例题：

1、实现字典树

2、查找字典树

```
package trie;
class TrieNode {
```

```java
// R links to node children
private TrieNode[] links;

private final int R = 26;

private boolean isEnd;

public TrieNode() {
    links = new TrieNode[R];
}

public boolean containsKey(char ch) {
    return links[ch -'a'] != null;
}
public TrieNode get(char ch) {
    return links[ch -'a'];
}
public void put(char ch, TrieNode node) {
    links[ch -'a'] = node;
}
public void setEnd() {
    isEnd = true;
}
public boolean isEnd() {
    return isEnd;
}

```

}

class Trie {

private TrieNode root;

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

// Inserts a word into the trie.
public void insert(String word) {
    TrieNode node = root;
```

```
    for (int i = 0; i < word.length(); i++) {
        char currentChar = word.charAt(i);
        if (!node.containsKey(currentChar)) {
            node.put(currentChar, new TrieNode());
        }
        node = node.get(currentChar);
    }
    node.setEnd();
}

```

private TrieNode searchPrefix(String word) {
TrieNode node = root;
for (int i = 0; i < word.length(); i++) {
char curLetter = word.charAt(i);
if (node.containsKey(curLetter)) {
node = node.get(curLetter);
} else {
return null;
}
}
return node;
}

```
// Returns if the word is in the trie.
public boolean search(String word) {
   TrieNode node = searchPrefix(word);
   return node != null && node.isEnd();
}

```

}

原文链接：https://blog.csdn.net/weixin_44625138/article/details/101149322