# 字典树

## 1、概述

**基本概念：**

字典树（Trie树），又称为前缀树（Prefix Tree）单词查找树或者键树，是一种多叉树结构。

a

b

c

e

f

r

g

图1-1 字典树

上图所示为一个由字符串{“acr”, ”ae”, ”acg”, ”bf” }组成，能看出啥规律吗？可以得出如下性质：

**基本性质：**

* 根节点不包含字符，除根节点外的每一个子节点都包含一个字符；
* 从根节点到某一节点。路径上经过的字符连接起来，就是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；

**应用场景：**

1. **字符串**
2. **词频统计**
3. **字符串排序**
4. **前缀匹配**
5. **作为辅助结构**

**优缺点：**

**优点：**

1. 插入和查询的效率很高，均是O（m），其中 m 是待插入/查询的 字符串长度 。

关于查询，有人会说hash表时间复杂度是O（1）不是更快？但是哈希搜索的效率取决于哈希函数的好坏，若一个坏的hash函数导致了很多冲突，效率不一定比Trie树高。

1. Trie树中不同的关键字不会产生冲突。
2. Trie树中只有在允许一个关键字关联多个值的情况下才有类似hash碰撞发生。
3. Trie树不用求hash值，对短字符串有更快的速度。通常，求hash值也是需要遍历字符串的(与hash函数相关)。
4. Trie树可以对关键字 按照字典序排序 （先序遍历）。

字典排序（lexicographical order）是一种对于随机变量形成序列的排序方法。其方法是，按照字母顺序，或者数字小大顺序，由小到大的形成序列。

1. 每一颗Trie树都可以被看做一个简单版的确定有限状态的自动机（DFA，deterministic finite automation），也就是说，对于一个任意给定属于该自动机的状态（①）和一个属于该自动机字母表的字符（②），都可以根据给定的转移函数（③）转到下一个状态。其中：

1、对于Trie树的每一个节点都确定一个自动机的状态。

2、给定一个属于该自动机字母表的字符，在图中可以看到根据不同字符形成的分支；

3、从当前节点进入下一层次节点的过程进过状态转移函数得出。

核心思想：**空间换时间，利用字符串的公共前缀来减少无谓的字符串比较以达到提高查询效率的目的。**

**缺点：**

1. **查询效率低于hash表**
2. **空间消耗大**

## 2、实现

**节点类：**

*/\*\*  
 \* 节点类  
 \*/*public class TrieNode {  
 private LinkedList<TrieNode> children;//子节点  
 private char data;//节点字符  
 private int freq;//频率  
 boolean isEnd;//是否是最后  
  
 public TrieNode(char data) {  
 this.data = data;  
 freq = 0;  
 isEnd = false;  
 this.children = new LinkedList<>();  
 }  
  
 */\*\*  
 \* 查找字符子节点  
 \** ***@param*** *c  
 \** ***@return*** *\*/* public TrieNode childNode(char c) {  
 if (null != children) {  
 for (TrieNode child : children) {  
 if (child.getData() == c) {  
 return child;  
 }  
 }  
 }  
 return null;  
 }

。。。  
}

主要包括四个属性：

1、children 存放所有子节点

2、data 存放该节点代表的字符

3、freq 表示当前字符的出现频率

4、isEnd表示是否为叶子节点

childNode方法查找c字符的子节点，没有返回null。

**字典树：**

package Trie;  
  
public class TrieTree {  
 private TrieNode root;  
  
 public TrieTree() {  
 this.root = new TrieNode(' ');  
 }  
  
 */\*\*  
 \* 插入节点  
 \*  
 \** ***@param*** *word  
 \*/* public void insert(String word) {  
 。。。  
 }  
  
 */\*\*  
 \* 删除  
 \** ***@param*** *word  
 \*/* public void remove(String word) {  
 。。。  
 }  
  
 */\*\*  
 \* 查找是否存在  
 \** ***@param*** *word  
 \** ***@return*** *\*/* public boolean search(String word) {  
 。。。  
 }  
  
 */\*\*  
 \* 查找返回节点  
 \** ***@param*** *word  
 \** ***@return*** *\*/* public TrieNode searchNode(String word) {  
 。。。  
 }  
  
 */\*\*  
 \* 获取词频  
 \** ***@param*** *word  
 \** ***@return*** *\*/* public int getFreq(String word) {  
 。。。  
 }  
}

**主要存在一个跟节点，表示空节点。**

**插入：**

public void insert(String word) {  
 TrieNode current = root;  
 for (int i = 0; i < word.length(); i++) {  
 TrieNode child = current.childNode(word.charAt(i));//插入字符，返回节点  
 if (null != child)//节点存在 将返回节点作为当前节点，  
 current = child;  
 else {//节点不存在 添加新节点 当前节点为新节点  
 current.getChildren().add(new TrieNode(word.charAt(i)));  
 current = current.childNode(word.charAt(i));  
 }  
 current.addFreq(1);//出现次数  
 }  
 current.setEnd(true);  
}

**查找**

public boolean search(String word) {  
 TrieNode current = root;  
 for (int i = 0; i < word.length(); i++) {//遍历所有字符  
 if (null == current.childNode(word.charAt(i)))//字符不存在就返回失败  
 return false;  
 else  
 current = current.childNode(word.charAt(i));//字符存在，就作为当前节点继续查找  
 }  
  
 if (current.isEnd())  
 return true;  
 else  
 return false;  
}  
public TrieNode searchNode(String word) {  
 TrieNode current = root;  
 for (int i = 0; i < word.length(); i++) {  
 if (null == current.childNode(word.charAt(i)))  
 return null;  
 else  
 current = current.childNode(word.charAt(i));  
 }  
  
 if (current.isEnd())  
 return current;  
 else  
 return null;  
}

**删除：**

public void remove(String word) {  
 if (!search(word))  
 return;  
  
 TrieNode current = root;  
 for (char c : word.toCharArray()) {  
 TrieNode child = current.childNode(c);//查找到子节点  
 if (child.getFreq() == 1) {  
 current.getChildren().remove(child);//子节点出现一次 直接删除  
 return;  
 }else{  
 child.subFreq(1);//子节点出现大于一次 频率减一，继续查找子节点  
 current = child;  
 }  
 }  
 current.setEnd(false);  
}

**频率：**

public int getFreq(String word) {  
 TrieNode trieNode = searchNode(word);  
 if(null != trieNode){  
 return trieNode.getFreq();  
 }else{  
 return 0;  
 }  
}