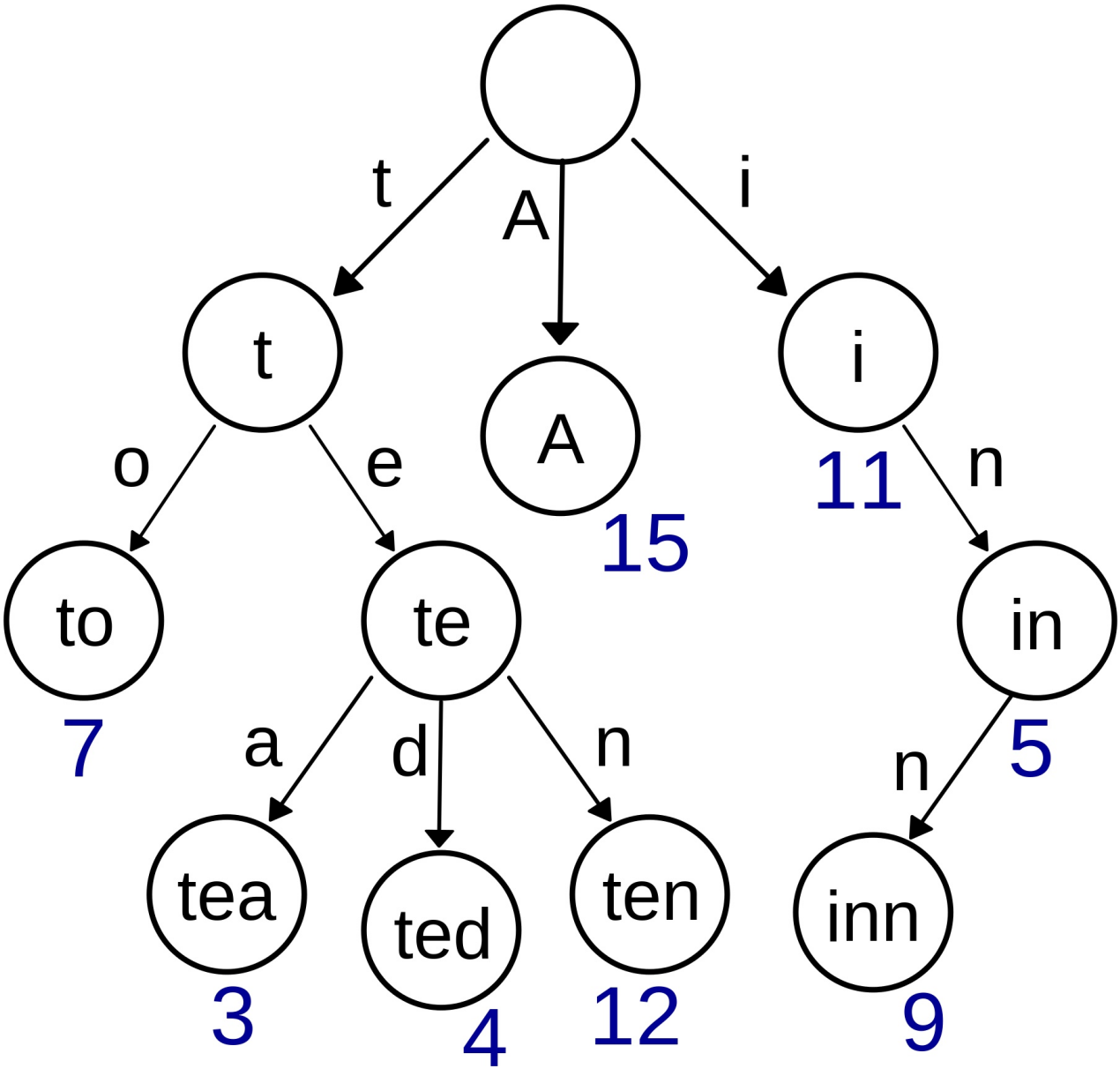
2019-06-03-Trie

定义

字典树是一种树形解构，利用字符串的公共前缀来节约存储空间，一个保存了8个键的trie结构，"A", "to", "tea", "ted", "ten", "i", "in", and "inn"如图所示



性质

字典树基本性质如下

1. 根节点没有路径字符，其余每一个节点都被一个字符路径找到
2. 从根节点到某一节点，将路径上经过的字符连起来为对应字符串

3. 每个节点引出的所有字符路径上的字符都不一样

搜索与添加过程

1. 从根节点开始搜索
2. 取得要查找的单词的第一个字母，选择对应字符路径向下搜索
3. 字符路径指向第二层节点上，根据第二个字母选择对应的字符继续搜索
4. 一直向下搜索，如果单词搜索完后，最后一个节点是终止节点，说明找到。如果最后一个节点不是终止节点，或者还没搜索完单词就没有后续节点了，说明这个单词没添加过，不断生成新的节点并添加即可。

实现

节点 TrieNode

path 代表有多少单词公用这个节点，end表示有多少单词以该节点结尾，map的key 代表该节点的字符路径，value 是字符路径指向的节点，在只有小写字母的情况下，map.length=26.

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

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

插入

设 str 长为n，从左到右遍历 str 中每个字符，并据此找下一个节点。如果找的过程中节点不存在，就新建节点，并令其 path = 1，如果存在，就令其 path++。通过最后一个字符找到最后一个节点时，令其path++，end++

```
public void insert(String word) {
        if (word == null) {
            return;
        }
        char[] chs = word.toCharArray();
        TrieNode node = root;
        node.path++;
        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++;
    }
```

搜索

依次遍历 str 中每个字符，并依次从头节点开始根据每一个 字符找下一个节点，如果节点不存在，说明没有添加进trie，返回 false，如果能通过 str[n-1]找到最后一个节点，记为e，若e.end!=0,说明有单词通过str[n-1]的字符路径，否则返回fasle。

```
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;
    }
```

删除

先调用 search() 函数，看 str 在不在 Trie 中，若不在则直接返回。否则和上边一样，从左到右遍历 str 中每个字符并寻找下一个节点。扫的过程中，经过的每一个节点，其 path-1，若发现某 path 减完后已经为0，就从当前节点map中删除后续所有路径，然后返回。如果扫到最后一个即诶单，记为e，令e.path--,e.end--.

```
public void delete(String word) {
        if (search(word)) {
            char[] chs = word.toCharArray();
            TrieNode node = root;
            node.path--;
            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--;
        }
    }
```

返回以 str 为前缀的单词数量

与查找操作类似，根据 pre 不断找到节点，返回最后的节点的 path 值即可。

```
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;
    }
```

特性

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，从上述实现中可以看出，插入查询的时间复杂度都与待操作字符串长度线性相关。

应用

1. 前缀匹配
   比如搜索引擎的自动补全
2. 字符串检索
3. 词频统计
4. 字符串排序