Trie树

简介

Trie树，又称单词查找树、字典树，是一种树形结构，是一种哈希树的变种，是一种用于快速检索的多叉树结构。也叫前缀树，是一种字符串的快速查找树，典型应用是用于统计和排序大量的字符串（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：最大限度地减少无谓的字符串比较，查询效率比哈希表高。

trie树的特点

1. 根节点不包含字符，除根节点意外每个节点只包含一个字符。
2. 从根节点到某一个节点，路径上经过的字符连接起来，为该节点对应的字符串。
3. 每个节点的所有子节点包含的字符串不相同。

Trie树有一些特性

1. 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。
2. 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。
3. 每个节点的所有子节点包含的字符都不相同。
4. 如果字符的种数为n，则每个结点的出度为n，这也是空间换时间的体现，浪费了很多的空间。
5. 插入查找的复杂度为O(n)，n为字符串长度。

基本思想（以字母树为例）

1. 插入过程

   对于一个单词，从根开始，沿着单词的各个字母所对应的树中的节点分支向下走，直到单词遍历完，将最后的节点标记为红色，表示该单词已插入Trie树。

2. 查询过程

   同样的，从根开始按照单词的字母顺序向下遍历trie树，一旦发现某个节点标记不存在或者单词遍历完成而最后的节点未标记为红色，则表示该单词不存在，若最后的节点标记为红色，表示该单词存在。

trie字典树的数据结构

利用串构建一个字典树，这个字典树保存了串的公共前缀信息，因此可以降低查询操作的复杂度。 下面以英文单词构建的字典树为例，这棵Trie树中每个结点包括26个孩子结点，因为总共有26个英文字母(假设单词都是小写字母组成)。 则可声明包含Trie树的结点信息的结构体:

```
typedef struct Trie_node { int count; // 统计单词前缀出现的次数 struct Trie_node* next[26];// 指向各个子树的指针 bool exist; // 标记该结点处是否构成单词 }TrieNode , *Trie;
```

如下图所示