

35|Trie树：如何实现搜索引擎的搜索关键词提示功能？

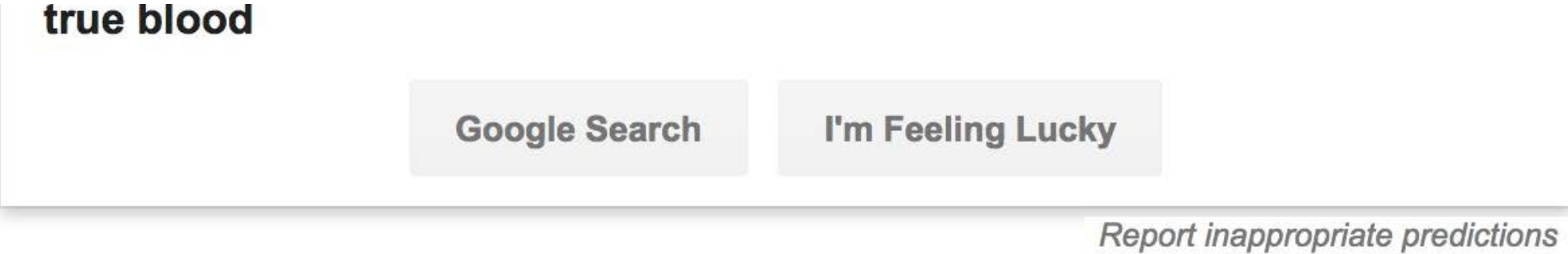
搜索引擎的搜索关键词提示功能，我想你应该不陌生吧？为了方便快速输入，当你在搜索引擎的搜索框中，输入要搜索的文字的某一部分的时候，搜索引擎就会自动弹出下拉框，里面是各种关键词提示。你可以直接从下拉框中选择你要搜索的东西，而不用把所有内容都输入进去，一定程度上节省了我们的搜索时间。



trie



trie **chofu**
trie **data structure**
trie **time complexity**
trie **utami**
trie **geeksforgeeks**
trie **data structure java**
trie **python**
trie **rack**
trie **vs tree**



尽管这个功能我们几乎天天在用，作为一名工程师，你是否思考过，它是怎么实现的呢？它底层使用的是哪种数据结构和算法呢？

像Google、百度这样的搜索引擎，它们的关键词提示功能非常全面和精准，肯定做了很多优化，但万变不离其宗，底层最基本的原理就是今天要讲的这种数据结构：Trie树。

什么是“Trie树”？

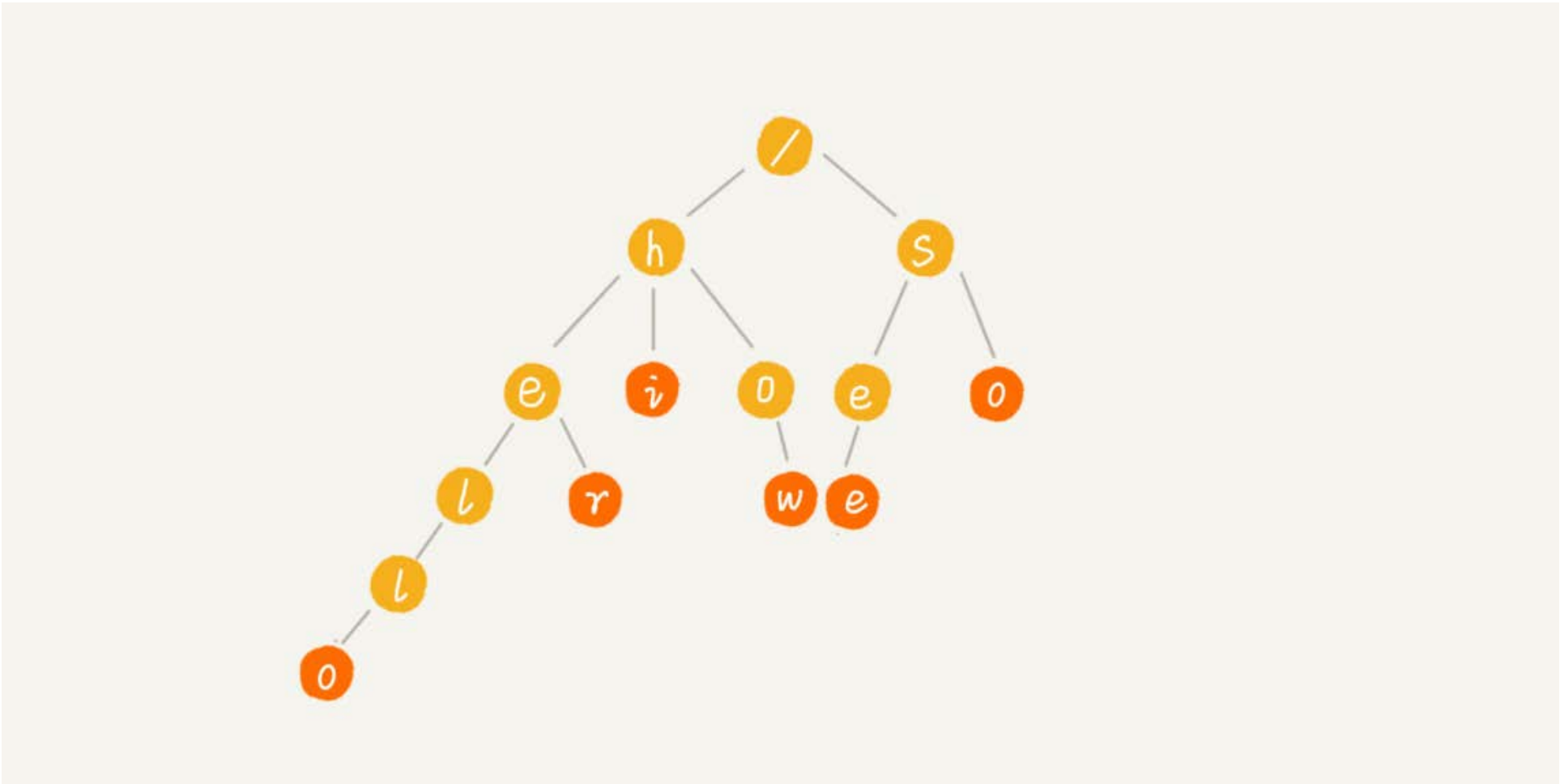
Trie树，也叫“字典树”。顾名思义，它是一个树形结构。它是一种专门处理字符串匹配的数据结构，用来解决在一组字符串集合中快速查找某个字符串的问题。

当然，这样一个问题可以有多种解决方法，比如散列表、红黑树，或者我们前面几节讲到的一些字符串匹配算法，但是，Trie树在这个问题的解决上，有它特有的优点。不仅如此，Trie树能解决的问题也不限于此，我们一会儿慢慢分析。

现在，我们先来看下，Trie树到底长什么样子。

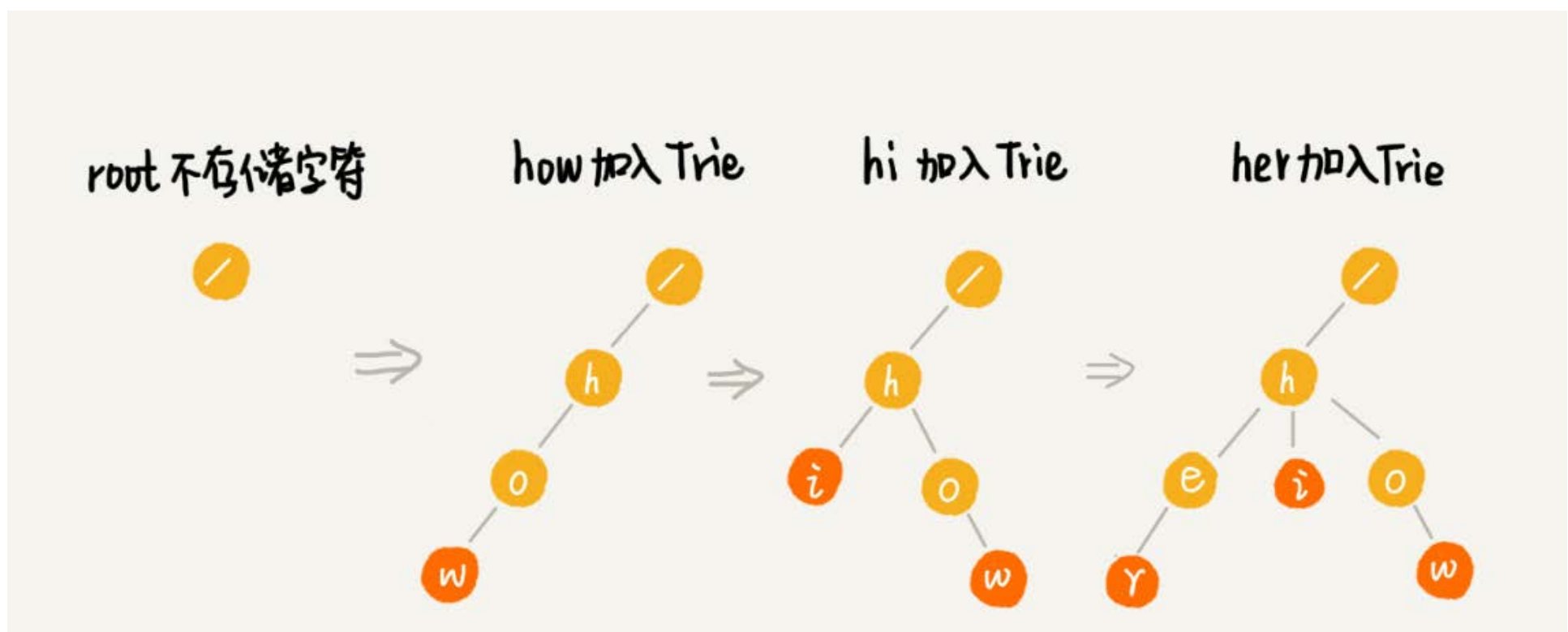
我举个简单的例子来说明一下。我们有6个字符串，它们分别是：how, hi, her, hello, so, see。我们希望在里面多次查找某个字符串是否存在。如果每次查找，都是拿要查找的字符串跟这6个字符串依次进行字符串匹配，那效率就比较低，有没有更高效的方法呢？

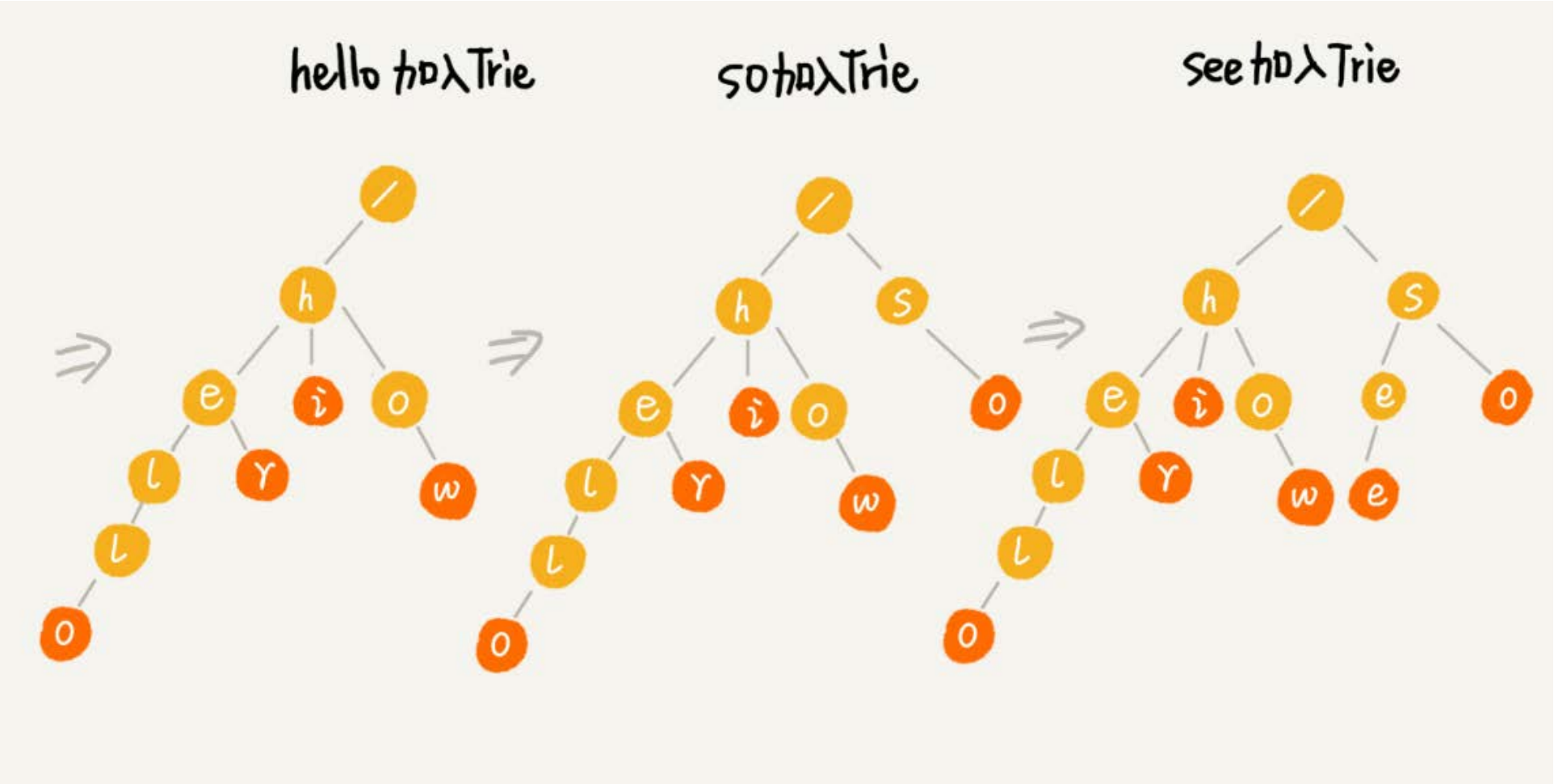
这个时候，我们就可以先对这6个字符串做一下预处理，组织成Trie树的结构，之后每次查找，都是在Trie树中进行匹配查找。Trie树的本质，就是利用字符串之间的公共前缀，将重复的前缀合并在一起。最后构造出来的就是下面这个图中的样子。



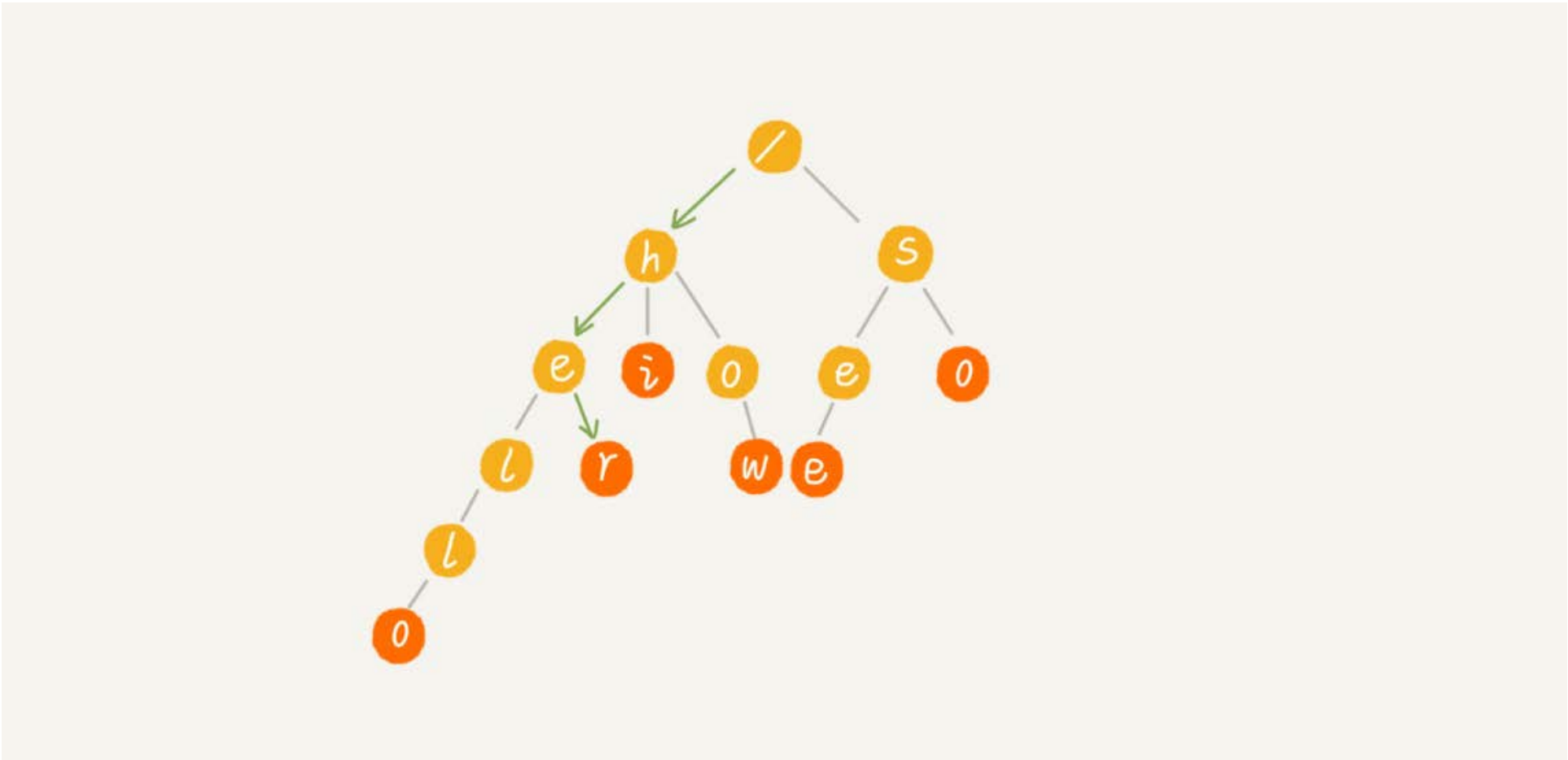
其中，根节点不包含任何信息。每个节点表示一个字符串中的字符，从根节点到红色节点的一条路径表示一个字符串（注意：红色节点并不都是叶子节点）。

为了让你更容易理解Trie树是怎么构造出来的，我画了一个Trie树构造的分解过程。构造过程的每一步，都相当于往Trie树中插入一个字符串。当所有字符串都插入完成之后，Trie树就构造好了。

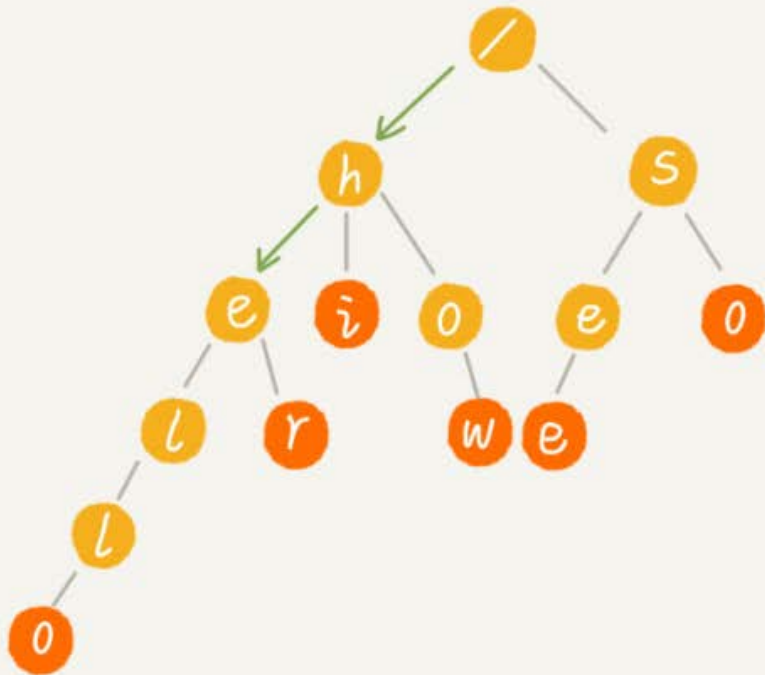




当我们在Trie树中查找一个字符串的时候，比如查找字符串“her”，那我们将要查找的字符串分割成单个的字符h，e，r，然后从Trie树的根节点开始匹配。如图所示，绿色的路径就是在Trie树中匹配的路径。



如果我们要查找的是字符串“he”呢？我们还用上面同样的方法，从根节点开始，沿着某条路径来匹配，如图所示，绿色的路径，是字符串“he”匹配的路径。但是，路径的最后一个节点“e”并不是红色的。也就是说，“he”是某个字符串的前缀子串，但并不能完全匹配任何字符串。



如何实现一棵Trie树？

知道了Trie树长什么样子，我们现在来看下，如何用代码来实现一个Trie树。

从刚刚Trie树的介绍来看，Trie树主要有两个操作，一个是将字符串集合构造成Trie树。这个过程分解开来的话，就是一个将字符串插入到Trie树的过程。另一个是在Trie树中查询一个字符串。

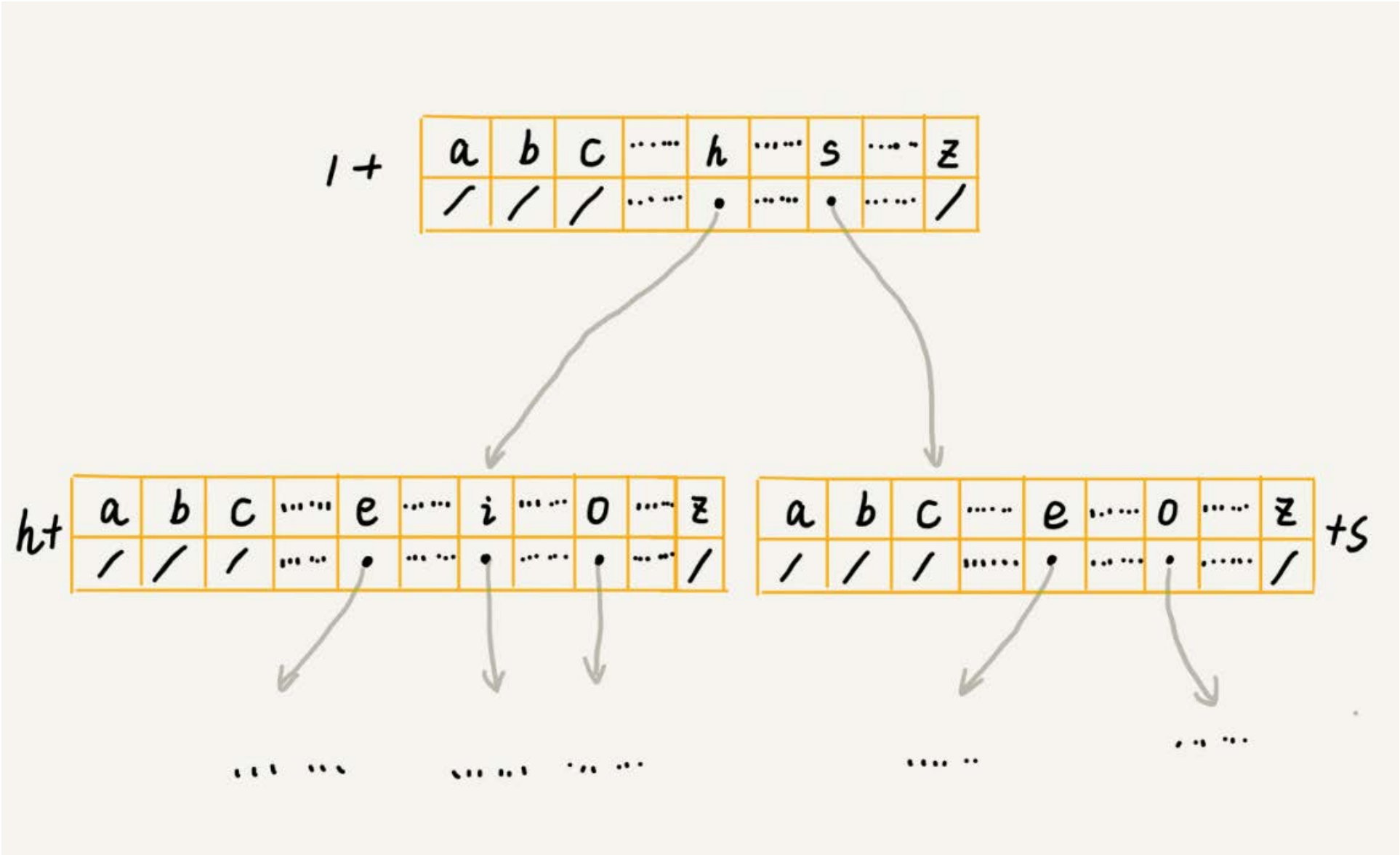
了解了Trie树的两个主要操作之后，我们再来看下，如何存储一个Trie树？

从前面的图中，我们可以看出，Trie树是一个多叉树。我们知道，二叉树中，一个节点的左右子节点是通过两个指针来存储的，如下所示Java代码。那对于多叉树来说，我们怎么存储一个节点的所有子节点的指针呢？

```
class BinaryTreeNode {
    char data;
    BinaryTreeNode left;
    BinaryTreeNode right;
}
```

我先介绍其中一种存储方式，也是经典的存储方式，大部分数据结构和算法书籍中都是这么讲的。还记得我们前面讲到的散列表吗？借助散列表的思想，我们通过一个下标与

字符——映射的数组，来存储子节点的指针。这句话稍微有点抽象，不怎么好懂，我画了一张图你可以看看。



假设我们的字符串中只有从a到z这26个小写字母，我们在数组中下标为0的位置，存储指向子节点a的指针，下标为1的位置存储指向子节点b的指针，以此类推，下标为25的位置，存储的是指向的子节点z的指针。如果某个字符的子节点不存在，我们就在对应的下标的位置存储null。

```
class TrieNode {
    char data;
    TrieNode children[26];
}
```


当我们在 树中查找字符串的时候，我们就可以通过字符的 码减去 的 码，迅速找到匹配的子节点的指针。比如， 的 码减去 的 码就是 ， 那子节点的指针就存储在数组中下标为3的位置中。

描述了这么多，有可能你还是有点懵，我把上面的描述翻译成了代码，你可以结合着一块看下，应该有助于你理解。

```
public class Trie {
    private TrieNode root = new TrieNode('/'); // 存储无意义字符

    // 往Trie树中插入一个字符串
    public void insert(char[] text) {
        TrieNode p = root;
        for (int i = 0; i < text.length; ++i) {
            int index = text[i] - 'a';
            if (p.children[index] == null) {
                TrieNode newNode = new TrieNode(text[i]);
                p.children[index] = newNode;
            }
            p = p.children[index];
        }
        p.isEndingChar = true;
    }

    // 在Trie树中查找一个字符串
    public boolean find(char[] pattern) {
        TrieNode p = root;
        for (int i = 0; i < pattern.length; ++i) {
            int index = pattern[i] - 'a';
            if (p.children[index] == null) {
                return false; // 不存在pattern
            }
            p = p.children[index];
        }
        if (p.isEndingChar == false) return false; // 不能完全匹配，只是前缀
        else return true; // 找到pattern
    }

    public class TrieNode {
        public char data;
        public TrieNode[] children = new TrieNode[26];
        public boolean isEndingChar = false;
        public TrieNode(char data) {
            this.data = data;
        }
    }
}
```

Trie树的实现，你现在应该搞懂了。现在，我们来看下，在**Trie**树中，查找某个字符串的时间复杂度是多少？

如果要在一组字符串中，频繁地查询某些字符串，用Trie树会非常高效。构建Trie树的过程，需要扫描所有的字符串，时间复杂度是 $O(n)$ （ n 表示所有字符串的长度和）。但是一旦构建成功之后，后续的查询操作会非常高效。

每次查询时，如果要查询的字符串长度是 k ，那我们只需要比对大约 k 个节点，就能完成查询操作。跟原本那组字符串的长度和个数没有任何关系。所以说，构建好Trie树后，在其中查找字符串的时间复杂度是 $O(k)$ ， k 表示要查找的字符串的长度。

Trie树真的很耗内存吗？

前面我们讲了Trie树的实现，也分析了时间复杂度。现在你应该知道，Trie树是一种非常独特的、高效的字符串匹配方法。但是，关于Trie树，你有没有听过这样一种说法：“Trie树是非常耗内存的，用的是一种空间换时间的思路”。这是什么原因呢？

刚刚我们在讲Trie树的实现的时候，讲到用数组来存储一个节点的子节点的指针。如果字符串中包含从a到z这26个字符，那每个节点都要存储一个长度为26的数组，并且每个数组存储一个8字节指针（或者是4字节，这个大小跟CPU、操作系统、编译器等有关）。而且，即便一个节点只有很少的子节点，远小于26个，比如3、4个，我们也要维护一

个长度为 2^6 的数组。

我们前面讲过，Trie树的本质是避免重复存储一组字符串的相同前缀子串，但是现在每个字符（对应一个节点）的存储远远大于1个字节。按照我们上面举的例子，数组长度为 2^6 ，每个元素是8字节，那每个节点就会额外需要 $2^6 * 8 = 208$ 个字节。而且这还是只包含 2^6 个字符的情况。

如果字符串中不仅包含小写字母，还包含大写字母、数字、甚至是中文，那需要的存储空间就更多了。所以，也就是说，在某些情况下，Trie树不一定会节省存储空间。在重复的前缀并不多的情况下，Trie树不但不能节省内存，还有可能会浪费更多的内存。

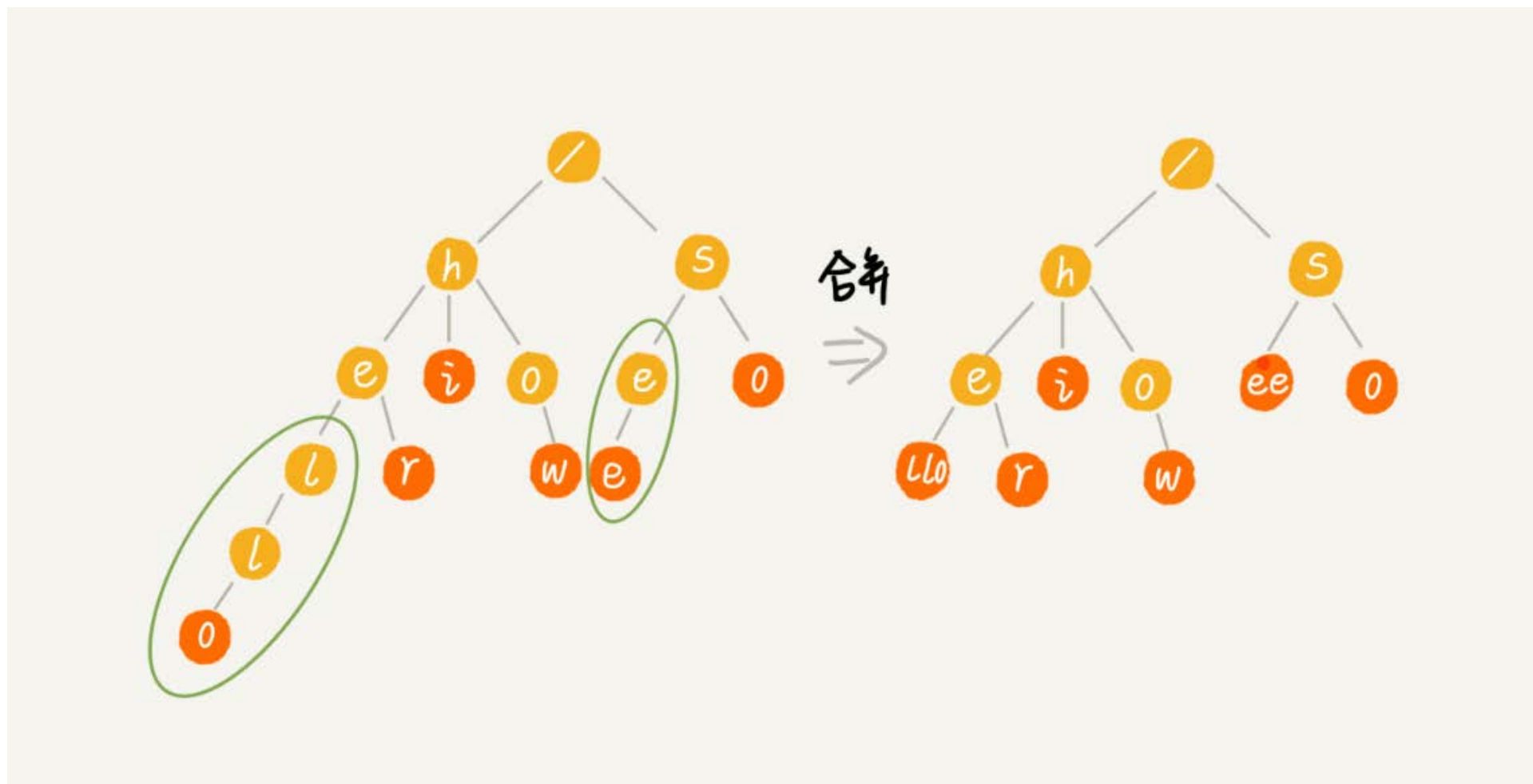
当然，我们不可否认，Trie树尽管有可能很浪费内存，但是确实非常高效。那为了解决这个内存问题，我们是否有其他办法呢？

我们可以稍微牺牲一点查询的效率，将每个节点中的数组换成其他数据结构，来存储一个节点的子节点指针。用哪种数据结构呢？我们的选择其实有很多，比如有序数组、跳表、散列表、红黑树等。

假设我们用有序数组，数组中的指针按照所指向的子节点中的字符的大小顺序排列。查询的时候，我们可以通过二分查找的方法，快速查找到某个字符应该匹配的子节点的指针。但是，在往Trie树中插入一个字符串的时候，我们为了维护数组中数据的有序性，就会稍微慢了点。

替换成其他数据结构的思路是类似的，这里我就不一一分析了，你可以结合前面学过的内容，自己分析一下。

实际上，Trie树的变体有很多，都可以在一定程度上解决内存消耗的问题。比如，缩点优化，就是对只有一个子节点的节点，而且此节点不是一个串的结束节点，可以将此节点与子节点合并。这样可以节省空间，但却增加了编码难度。这里我就不展开详细讲解了，你如果感兴趣，可以自行研究下。



Trie树与散列表、红黑树的比较

实际上，字符串的匹配问题，笼统上讲，其实就是数据的查找问题。对于支持动态数据高效操作的数据结构，我们前面已经讲过好多了，比如散列表、红黑树、跳表等等。实际上，这些数据结构也可以实现在一组字符串中查找字符串的功能。我们选了两组数据结构，散列表和红黑树，跟Trie树比较一下，看看它们各自的优缺点和应用场景。

在刚刚讲的这个场景，在一组字符串中查找字符串，Trie树实际上表现得并不好。它对要处理的字符串有及其严苛的要求。

第一，字符串中包含的字符集不能太大。我们前面讲到，如果字符集太大，那存储空间可能会浪费很多。即便可以优化，但也要付出牺牲查询、插入效率的代价。

第二，要求字符串的前缀重合比较多，不然空间消耗会变大很多。

第三，如果要用Trie树解决问题，那我们就要自己从零开始实现一个Trie树，还要保证没有bug，这个在工程上是将简单问题复杂化，除非必须，一般不建议这样做。

第四，我们知道，通过指针串起来的数据块是不连续的，而Trie树中用到了指针，所以，对缓存并不友好，性能上会打个折扣。

综合这几点，针对在一组字符串中查找字符串的问题，我们在工程中，更倾向于用散列表或者红黑树。因为这两种数据结构，我们都不需要自己去实现，直接利用编程语言中提供的现成类库就行了。

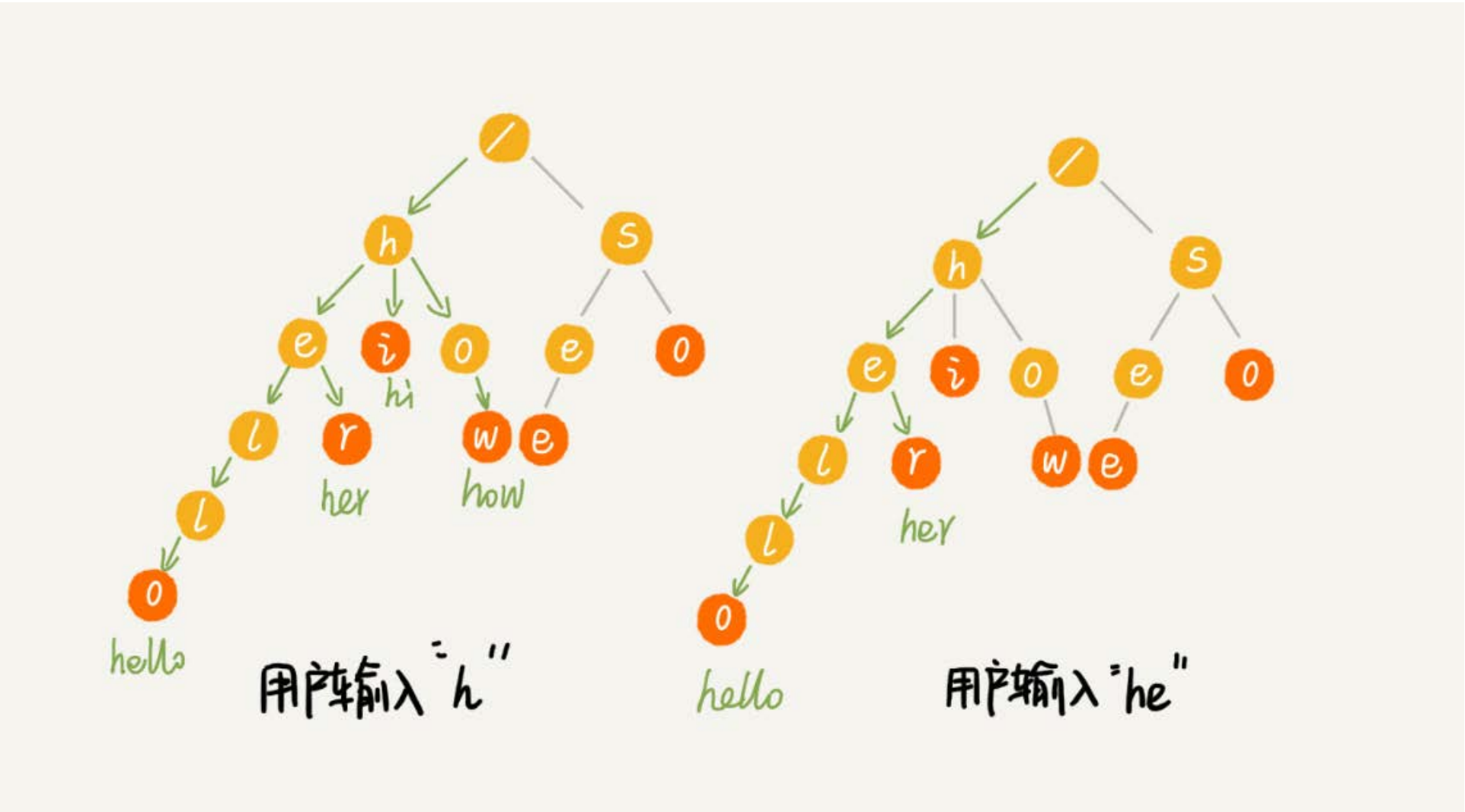
讲到这里，你可能要疑惑了，讲了半天，我对Trie树一通否定，还让你用红黑树或者散列表，那Trie树是不是就没用了呢？是不是今天的内容就白学了呢？

实际上，Trie树只是不适合精确匹配查找，这种问题更适合用散列表或者红黑树来解决。Trie树比较适合的是查找前缀匹配的字符串，也就是类似开篇问题的那种场景。

解答开篇

Trie树就讲完了，我们来看下开篇提到的问题：如何利用Trie树，实现搜索关键词的提示功能？

我们假设关键词库由用户的热门搜索关键词组成。我们将这个词库构建成一个Trie树。当用户输入其中某个单词的时候，把这个词作为一个前缀子串在Trie树中匹配。为了解方便，我们假设词库里只有hello、her、hi、how、so、see这6个关键词。当用户输入了字母h的时候，我们就把以h为前缀的hello、her、hi、how展示在搜索提示框内。当用户继续键入字母e的时候，我们就把以he为前缀的hello、her展示在搜索提示框内。这就是搜索关键词提示的最基本的算法原理。



不过，我讲的只是最基本的实现原理，实际上，搜索引擎的搜索关键词提示功能远非我讲的这么简单。如果再稍微深入一点，你就会想到，上面的解决办法遇到下面几个问题：

- 我刚讲的思路是针对英文的搜索关键词提示，对于更加复杂的中文来说，词库中的数据又该如何构建成Trie树呢？
- 如果词库中有很多关键词，在搜索提示的时候，用户输入关键词，作为前缀在Trie树中可以匹配的关键词也有很多，如何选择展示哪些内容呢？
- 像Google这样的搜索引擎，用户单词拼写错误的情况下，Google还是可以使用正确的拼写来做关键词提示，这个又是怎么做到的呢？

你可以先思考一下如何解决，如果不会也没关系，这些问题，我们会在实战篇里具体来讲解。

实际上，Trie树的这个应用可以扩展到更加广泛的一个应用上，就是自动输入补全，比如输入法自动补全功能、IDE代码编辑器自动补全功能、浏览器网址输入的自动补全功能等等。

内容小结

今天我们讲了一种特殊的树，Trie树。Trie树是一种解决字符串快速匹配问题的数据结构。如果用来构建Trie树的这一组字符串中，前缀重复的情况不是很多，那Trie树这种数据结构总体上来讲是比较费内存的，是一种空间换时间的解决问题思路。

尽管比较耗费内存，但是对内存不敏感或者内存消耗在接受范围内的情况下，在Trie树中做字符串匹配还是非常高效的，时间复杂度是 $O(k)$ ， k 表示要匹配的字符串的长度。

但是，Trie树的优势并不在于，用它来做动态集合数据的查找，因为，这个工作完全可以用更加合适的散列表或者红黑树来替代。Trie树最有优势的是查找前缀匹配的字符串，比如搜索引擎中的关键词提示功能这个场景，就比较适合用它来解决，也是Trie树比较经典的应用场景。

课后思考

我们今天有讲到，Trie树应用场合对数据要求比较苛刻，比如字符串的字符集不能太大，前缀重合比较多等。如果现在给你一个很大的字符串集合，比如包含1万条记录，如何通过编程量化分析这组字符串集合是否比较适合用Trie树解决呢？也就是如何统计字符串的字符集大小，以及前缀重合的程度呢？

欢迎留言和我分享，也欢迎点击“请朋友读”，把今天的内容分享给你的好友，和他一起讨论、学习。



数据结构与算法之美

为工程师量身打造的数据结构与算法私教课

王争

前 Google 工程师



新版升级：点击「👤请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

精选留言：

- ban 2018-12-13 22:29:36
上面代码里：p.isEndingChar = true; 应该是放在for循环的外面吧？
不然如果hello， 那不就变成 h e l l o 都是叶子节点？ [15赞]
- Smallfly 2018-12-12 17:44:09
思考题：

35|Trie树：如何实现搜索引擎的搜索关键词提示功能？

依次读取每个字符串的字符构建 Trie 树，用散列表来存储每一个节点。每一层树的所有散列表的元素用一个链表串联起来，求某一长度的前缀重合，在对应树层级上遍历该层链表，求链表长度，除以字符集大小，值越小前缀重合率越高。

遍历所有树层级的链表，存入散列表，最后散列表包含元素的个数，就代表字符集的大小。 [7赞]

• 传说中的成大大 2018-12-13 22:16:26

今天的课程比上两节的课程理解起来容易多了 总体觉得就像是构造出来的多叉树，相同的前缀字符串就是同一棵树下来的不同分之 [5赞]

• Jerry银银 2018-12-28 19:30:41

找到了一个Trie树的开源库：Apache Commons，里面有关于Trie的实现 [4赞]

作者回复2019-01-02 16:38:04

• kepmov 2018-12-12 00:27:13

trie树实际项目中由于内存问题用的不是很多，老师可以讲解下DAT（双数组trie树）的具体实现吗 [4赞]

作者回复2018-12-13 10:09:45

这个还是自己研究吧 内容太多了 文章有限。或者后面我收集下 统一写几篇加餐文章吧

• 夏洛克的救赎 2018-12-12 08:46:47

问题思考：

- 1 中文转换成ASCII码？
- 2 根据以往用户搜索记录，选择占比最高的
- 3 从词库检索匹配？ [3赞]

• 王鸿运 2018-12-29 09:44:05

isEndingChar可以修改成uint型字段，这样不仅能够判断是否包含该字符串，还可以进行字符串出现次数 [2赞]

• ZX 2018-12-17 20:55:47

老师，字符串匹配这里，还差后缀树没讲，很多场合需要用到这种结构，希望老师可以讲一讲 [2赞]

作者回复2018-12-18 09:44:04

那个更高级 不讲了 自学吧亲

• feifei 2018-12-16 10:37:23

如何统计字符串的字符集大小，以及前缀重合的程度呢？

统计字符集的大小，这个问题，其实就是在求字符的最小值以及最大值的问题。

我的解决办法

35|Trie树：如何实现搜索引擎的搜索关键词提示功能？

- 1，遍历字符串集合
- 2，将每个字符转化为int数字
- 3，设置最小以及最大的变量，当字符中比最大字符的变量大的时候，将最大字符变量改为当前字符，或者比最小字符小，就修改最小字符
- 4，遍历完成后，所求得的最大值与最小值的差，就是字符集的大小

前缀重合的程度，这个问题的求解，其实就是做字符的统计问题

我的解决办法：

使用哈希表来记录下统计数，key为字符，value为统计计数

遍历每条记录，假如每条记录中仅包含一个单词（如果多单词，多一步分隔操作，分隔成一个一个的单词）

统计计数算法，就是从前到后遍历，遇到存在的，加1，不存在，则存入hash表

比如hello这个单词，在哈希表中存储就为

h 1
he 1
hel 1
hell 1
hello 1

当再将出现，比如he
就会变成
h 2
he 2
hel 1
hell 1
hello 1

统计数据完成后，对这个结果计算重合的字符数与整个字符的占比，

具体计算公式为: count(value > 1) / count(all)

但我的算法复杂度有点高，是m*n,m表示整个字符的长度，n表示单个单表的长度。

[2赞]

- 等风来 2018-12-12 11:32:47
if (p.isEndingChar == false) return false; // 不能完全匹配，只是前缀
else return true; // 找到 pattern
这小段代码有点不大牛.^_
return p.isEndingChar;就好了 [2赞]

35|Trie树：如何实现搜索引擎的搜索关键词提示功能？

作者回复2018-12-12 14:39:31

嗯嗯 怕看不懂嘛

• Jerry银银 2018-12-28 11:59:03

老师帮忙推荐一些包含Trie树实现的优秀的开源库呗，好让我们深入研究 [1赞]

• 起点·终站 2018-12-17 17:20:22

看完后发现我们项目的屏蔽字检测就是用trie树写的。。666 [1赞]

• 传说中的成大大 2018-12-13 22:22:49

不过代码有点不太明白的地方在于 isendingchar是哪里来的？主要是干啥的呢？ [1赞]

• 您的好友William 2018-12-12 15:59:22

我来提供一个异想天开的想法哈哈，我想的是通过leetcode127题的Word Ladder的类似的方法实现，通过逐个改变每个字符串的后缀看看能不能匹配到其他字符串，进行BFS看看得改多少次才能匹配到原来的list里面有的单词，如果平均改的次数很多证明这个前缀不起什么作用，如果改的很少证明前缀的重复利用率很高！ [1赞]

• Michael 2019-01-24 22:41:09

前两节听得头皮发麻，这节至少听明白了，Trie树比较适合前缀字符串匹配，对于精确匹配，字符集比较大的场景，可以改用红黑树，散列表

• 不成熟的萌 2019-01-19 18:08:39

那么就要给出和前缀匹配的所有单词。

```
```cpp
// 现在我们利用startWith(prefix)来到了这个前缀节点，此时，我们要输出所有关键词，
// 即我们的目的是找到以当前前缀最后一个字母为根节点展开的所有单词
// R 为字典库大小
void collect(TrieNode *p, string prefix, queue<string> &q)
{
 if(nullptr == p)
 return;
 if(p->isEnding) q.push(prefix);
 for(int i = 0; i < R; ++i)
 {
 string tmp = prefix.append(i + 'a');
 collect(p->children[i], tmp, q);
 }
}
```

    q
```

35|Trie树：如何实现搜索引擎的搜索关键词提示功能？

最后 中存储的就是所有关键词。

这个方法在橙书的字符查找树中有实现。

- gopherliu 2019-01-17 20:05:27

以太坊好像用到了这种树状结构

- 黄世仁 2019-01-14 17:52:12

按照老师的讲解，实现了一个带自动补齐的TrieTree, <https://github.com/email2liyang/algo/commit/f3d4afe59aac7ee6b2031f392825445f656e50bc>

- qinggeouye 2019-01-14 01:04:32

Trie 树观察：

多叉树，根结点下的各分支之间，不存在前缀重合；

对于根结点下的每个分支，在同级的子节点，它们的父节点是公共前缀的结尾字符；

叶子节点的总数小于等于字符串的总数（如 he、her、hey、hello）；

求字符集，可以利用 **BFS** 遍历所有节点，存入哈希表；

- lm 2019-01-09 12:16:53

这种树结构是不是匹配到前缀后还得继续遍历前缀的子节点？这样提示字符串才能显示全

作者回复2019-01-10 10:12:13

是的