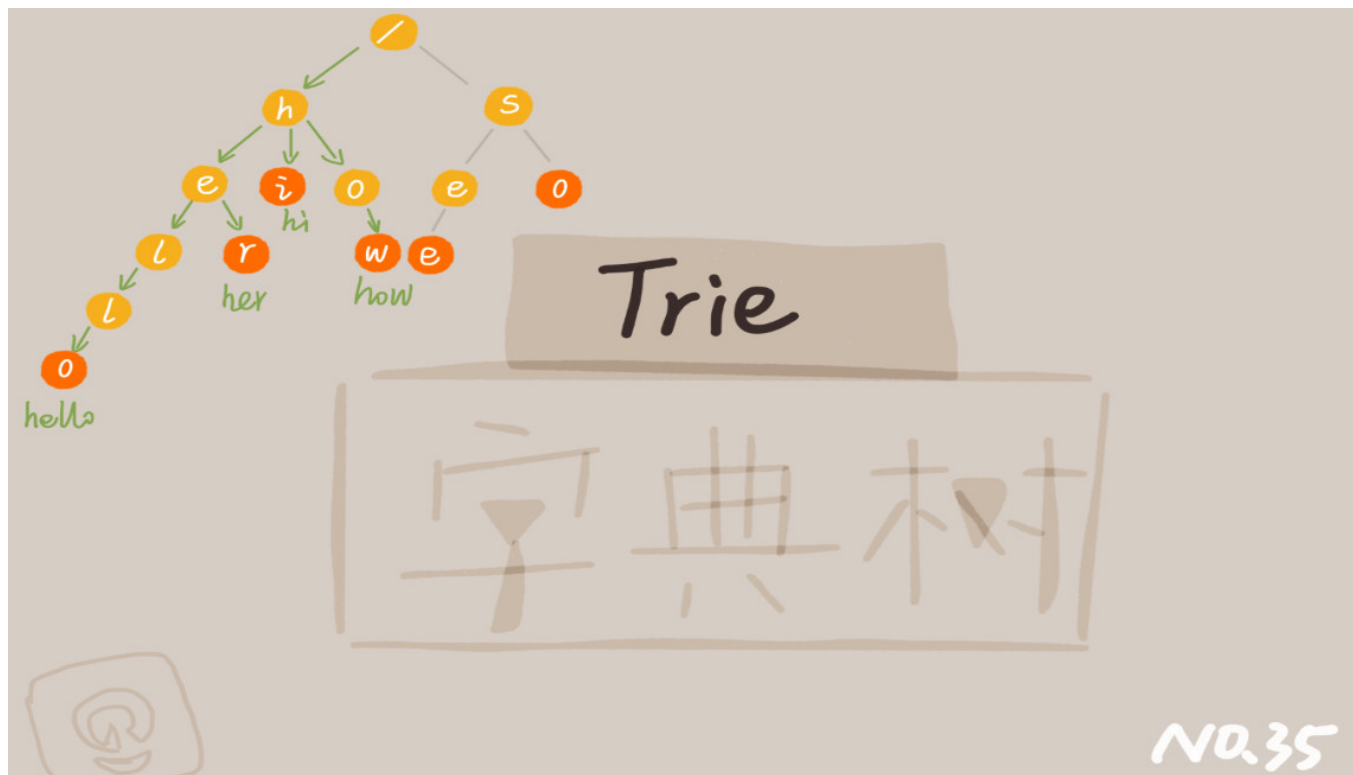


讲堂 > 数据结构与算法之美 > 文章详情

## 35 | Trie树：如何实现搜索引擎的搜索关键词提示功能？

2018-12-12 王争




### 35 | Trie树：如何实现搜索引擎的搜索关键词提示功能？

朗读人：修阳 14'32" | 9.99M

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**

trie **true blood**

Google Search

I'm Feeling Lucky

[Report inappropriate predictions](#)

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”？

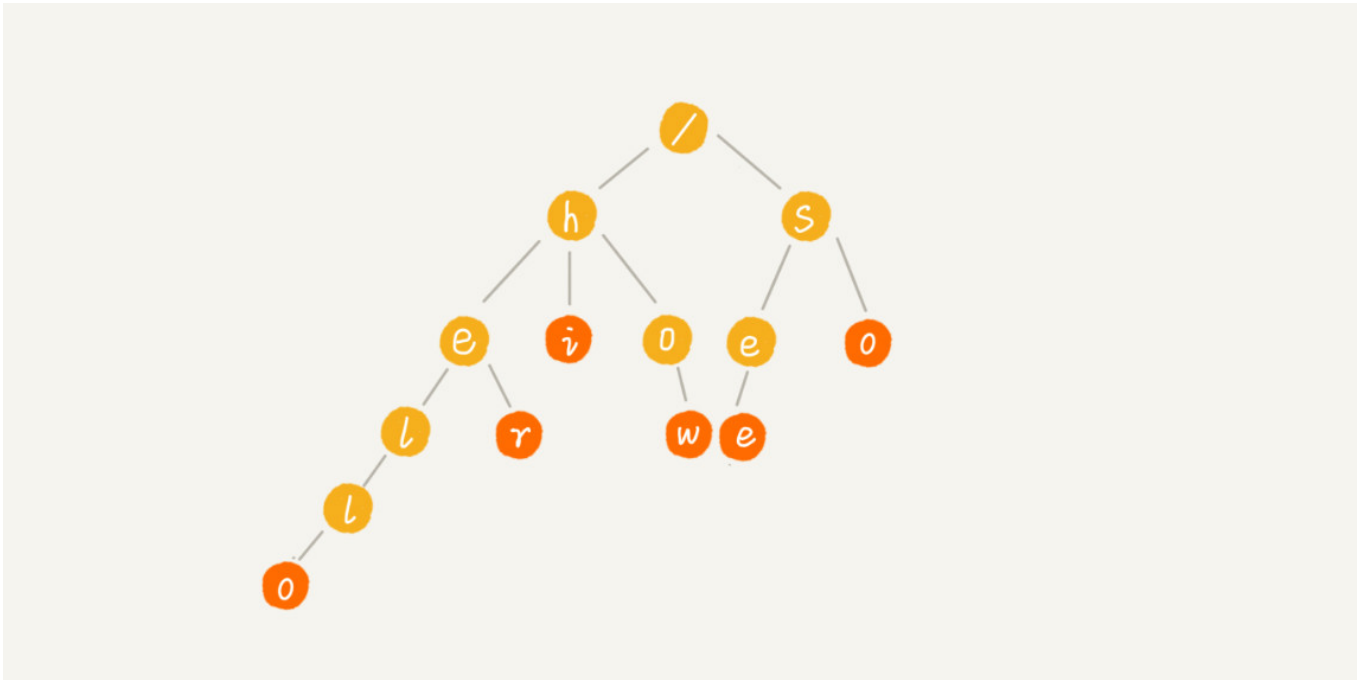
Trie 树，也叫“字典树”。顾名思义，它是一个树形结构。它是一种专门处理字符串匹配的数据结构，用来解决在一组字符串集合中快速查找某个字符串的问题。

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。

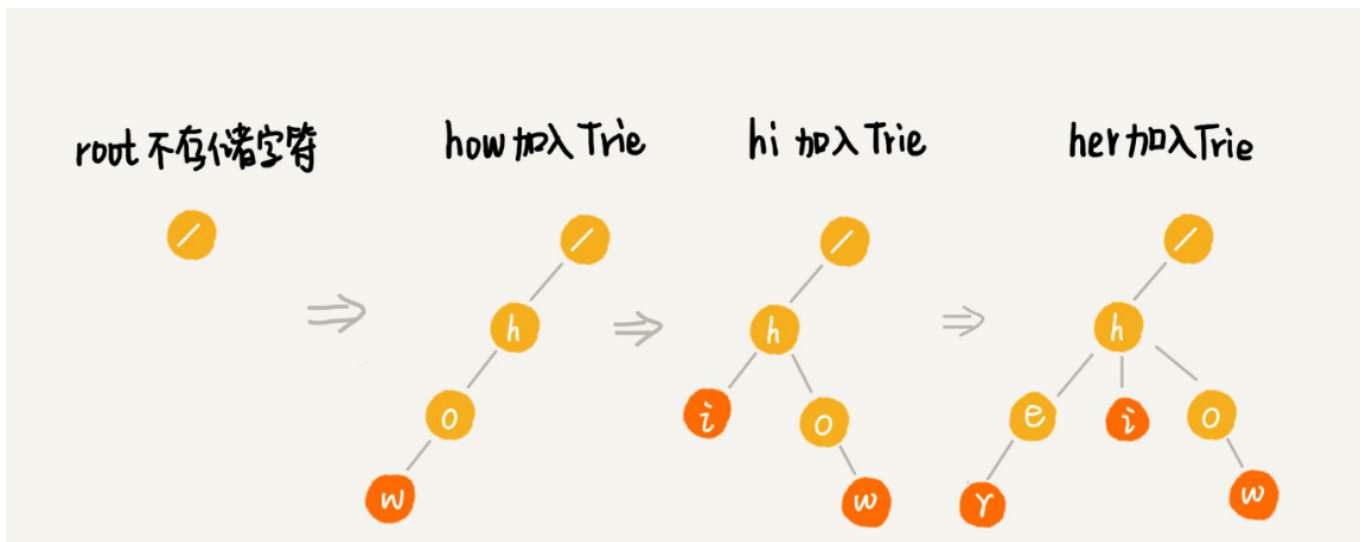
我举个简单的例子来说明一下。我们有 6 个字符串，它们分别是：how, hi, her, hello, so, see。我们希望在里面多次查找某个字符串是否存在。如果每次查找，都是拿要查找的字符串跟这 6 个字符串依次进行字符串匹配，那效率就比较低，有没有更高效的方法呢？

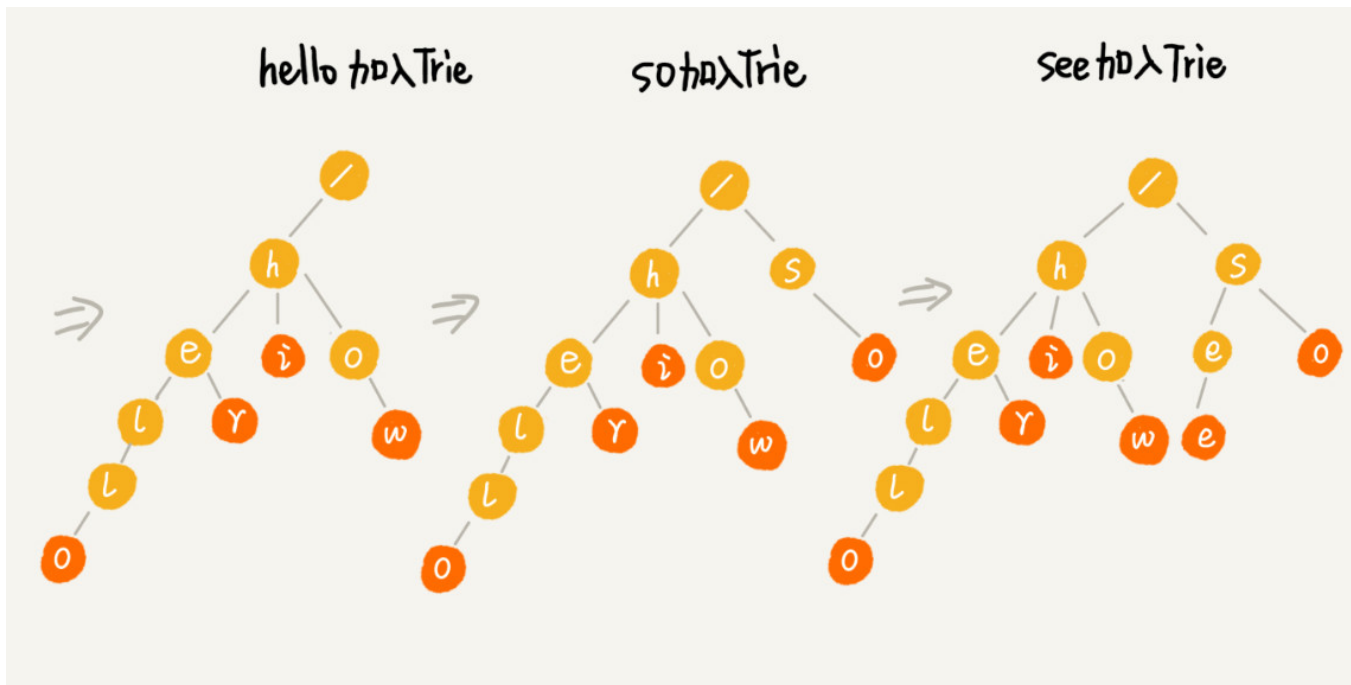
这个时候，我们就可以先对这 6 个字符串做一下预处理，组织成 Trie 树的结构，之后每次查找，都是在 Trie 树中进行匹配查找。**Trie 树的本质，就是利用字符串之间的公共前缀，将重复的前缀合并在一起。**最后构造出来的就是下面这个图中的样子。



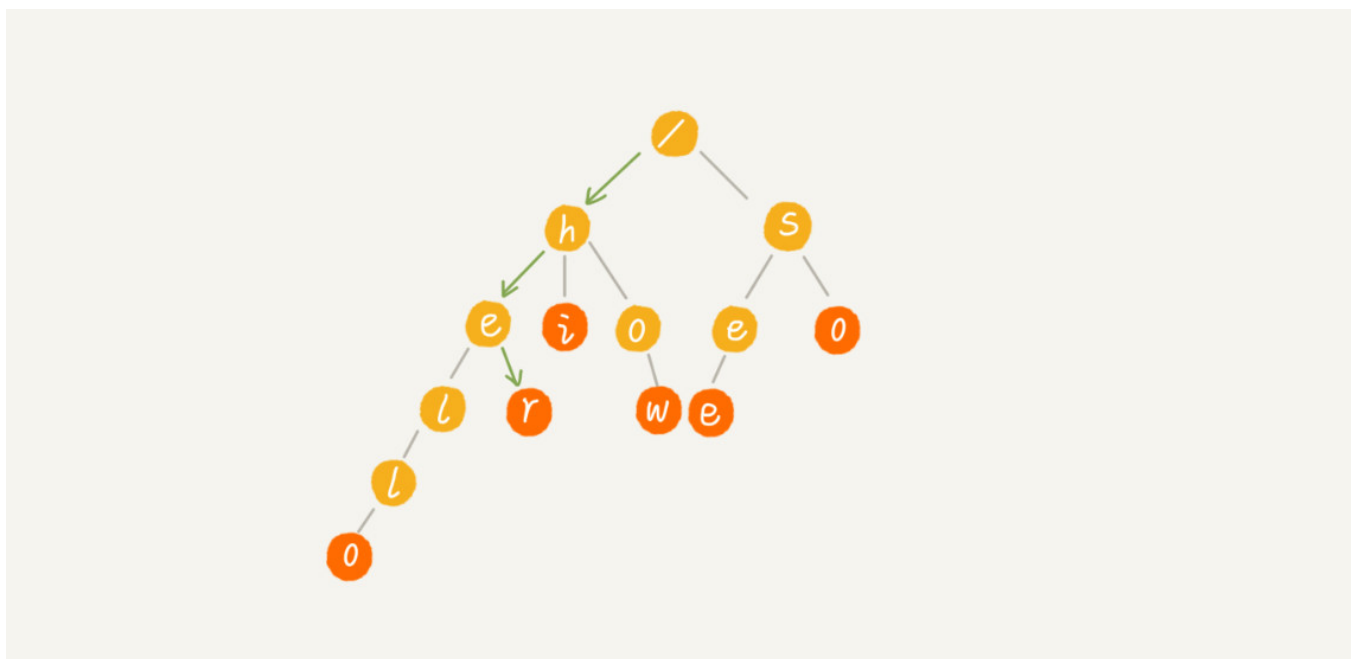
其中，根节点不包含任何信息。每个节点表示一个字符串中的字符，从根节点到红色节点的一条路径表示一个字符串（注意：红色节点并不都是叶子节点）。

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。

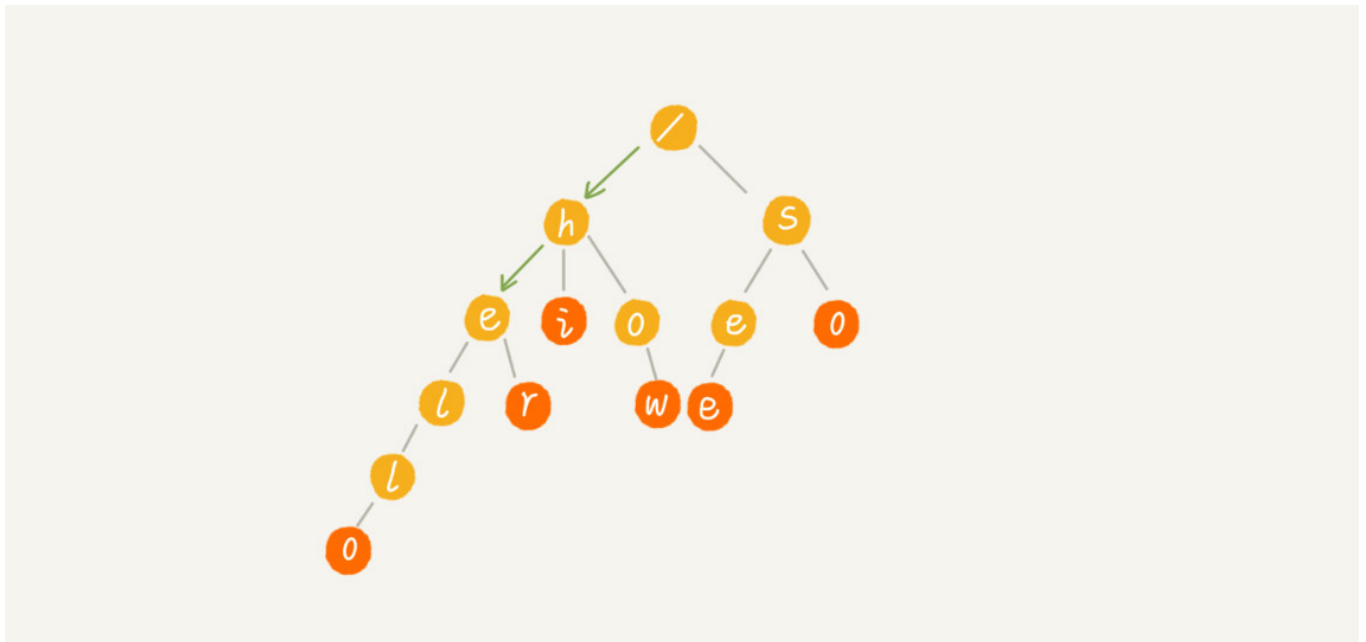




当我们在 Trie 树中查找一个字符串的时候，比如查找字符串“her”，那我们将要查找的字符串分割成单个的字符 h, e, r，然后从 Trie 树的根节点开始匹配。如图所示，绿色的路径就是在 Trie 树中匹配的路径。



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。

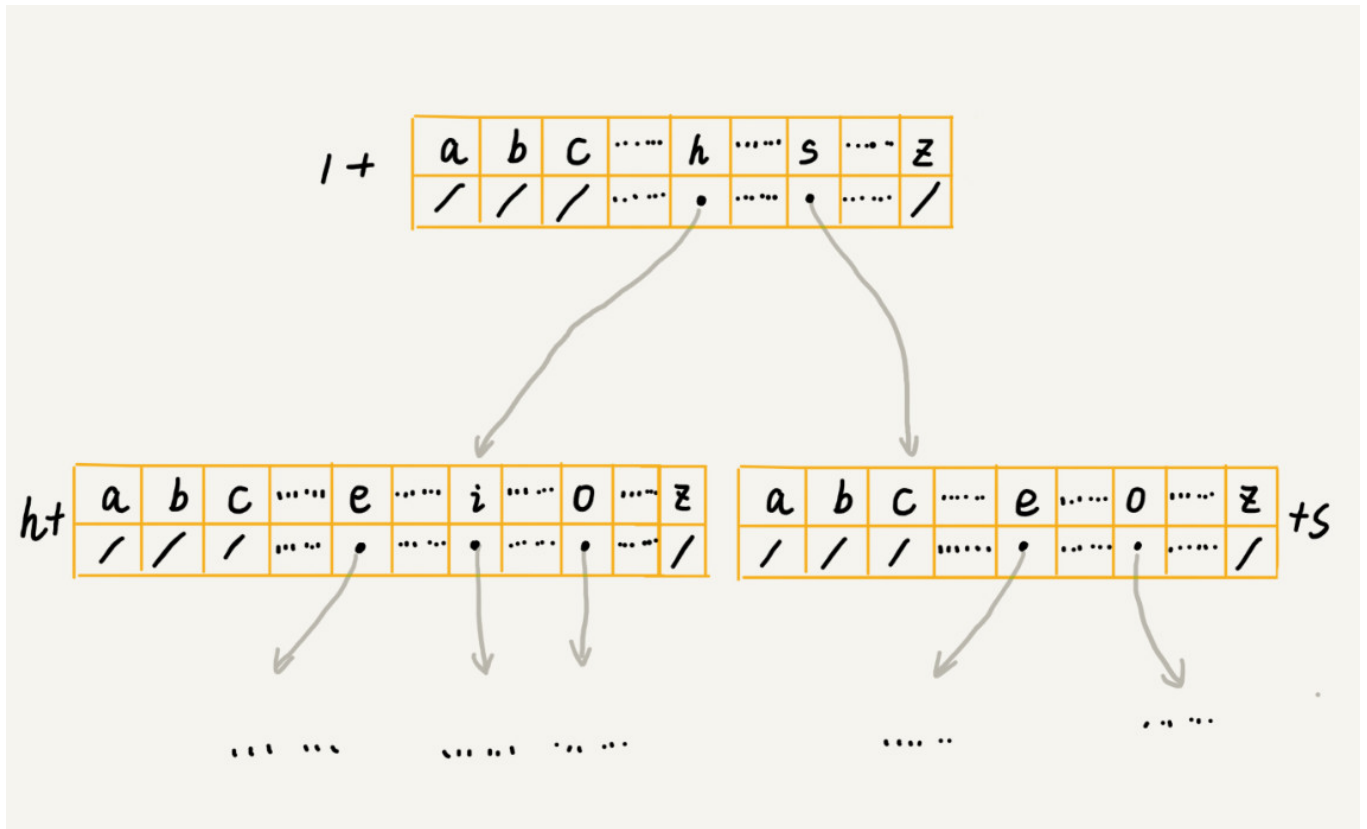
了解了 Trie 树的两个主要操作之后，我们再来看下，如何存储一个 Trie 树？

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？

```
1 class BinaryTreeNode {
2     char data;
3     BinaryTreeNode left;
4     BinaryTreeNode right;
5 }
```

[复制代码](#)

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一一映射的数组，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。

```
1 class TrieNode {
2     char data;
3     TrieNode children[26];
4 }
```

[复制代码](#)

当我们在 Trie 树中查找字符串的时候，我们就可以通过字符的 ASCII 码减去“a”的 ASCII 码，迅速找到匹配的子节点的指针。比如，d 的 ASCII 码减去 a 的 ASCII 码就是 3，那子节点 d 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。

```
1 public class Trie {
2     private TrieNode root = new TrieNode('/'); // 存储无意义字符
3
4     // 往 Trie 树中插入一个字符串
5     public void insert(char[] text) {
6         TrieNode p = root;
7         for (int i = 0; i < text.length; ++i) {
8
9             int index = text[i] - 'a';
```

[复制代码](#)

```
9         if (p.children[index] == null) {
10             TrieNode newNode = new TrieNode(text[i]);
11             p.children[index] = newNode;
12         }
13         p = p.children[index];
14     }
15     p.isEndingChar = true;
16 }
17
18 // 在 Trie 树中查找一个字符串
19 public boolean find(char[] pattern) {
20     TrieNode p = root;
21     for (int i = 0; i < pattern.length; ++i) {
22         int index = pattern[i] - 'a';
23         if (p.children[index] == null) {
24             return false; // 不存在 pattern
25         }
26         p = p.children[index];
27     }
28     if (p.isEndingChar == false) return false; // 不能完全匹配, 只是前缀
29     else return true; // 找到 pattern
30 }
31
32 public class TrieNode {
33     public char data;
34     public TrieNode[] children = new TrieNode[26];
35     public boolean isEndingChar = false;
36     public TrieNode(char data) {
37         this.data = data;
38     }
39 }
40 }
```

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”。这是为什么呢？

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 26 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 26，每个元素是 8 字节，那每个节点就会额外需要  $26 \times 8 = 208$  个字节。而且这还是只包含 26 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？

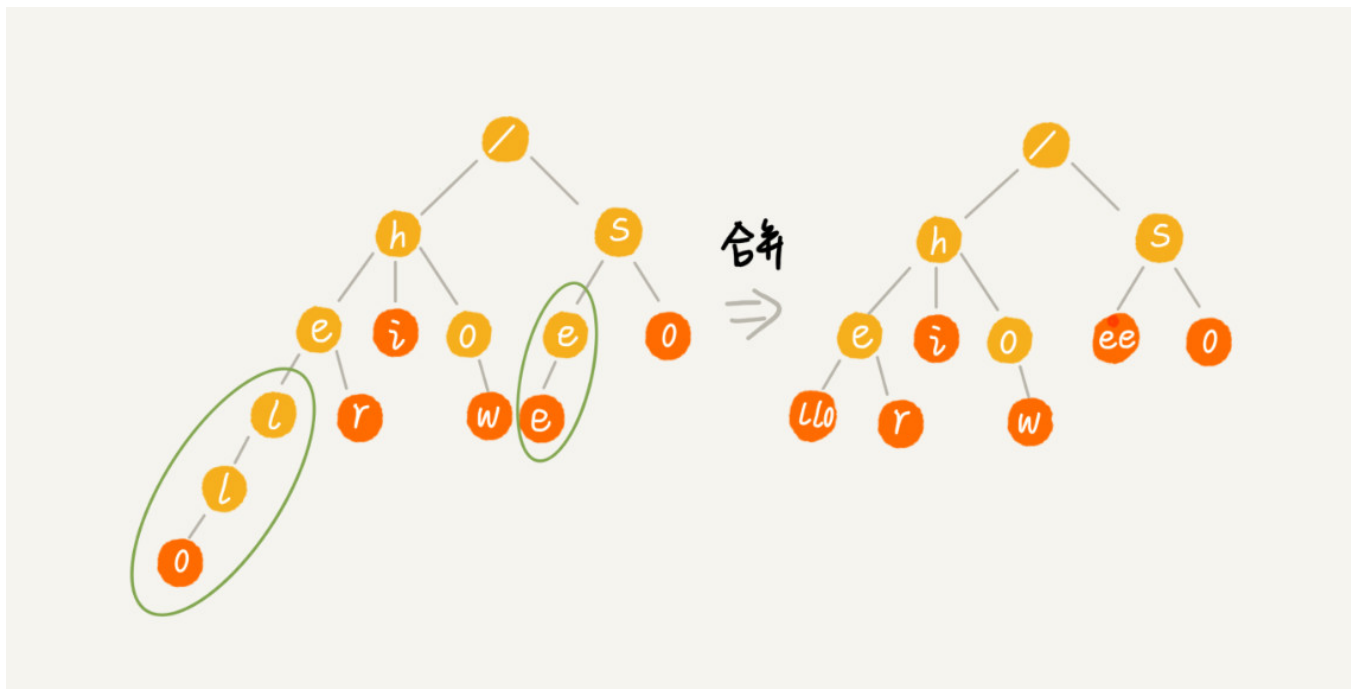
我们可以稍微牺牲一点查询的效率，将每个节点中的数组换成其他数据结构，来存储一个节点的子节点指针。用哪种数据结构呢？我们的选择其实有很多，比如有序数组、跳表、散列表、红黑树等。

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。我们选了两种数据结构，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。

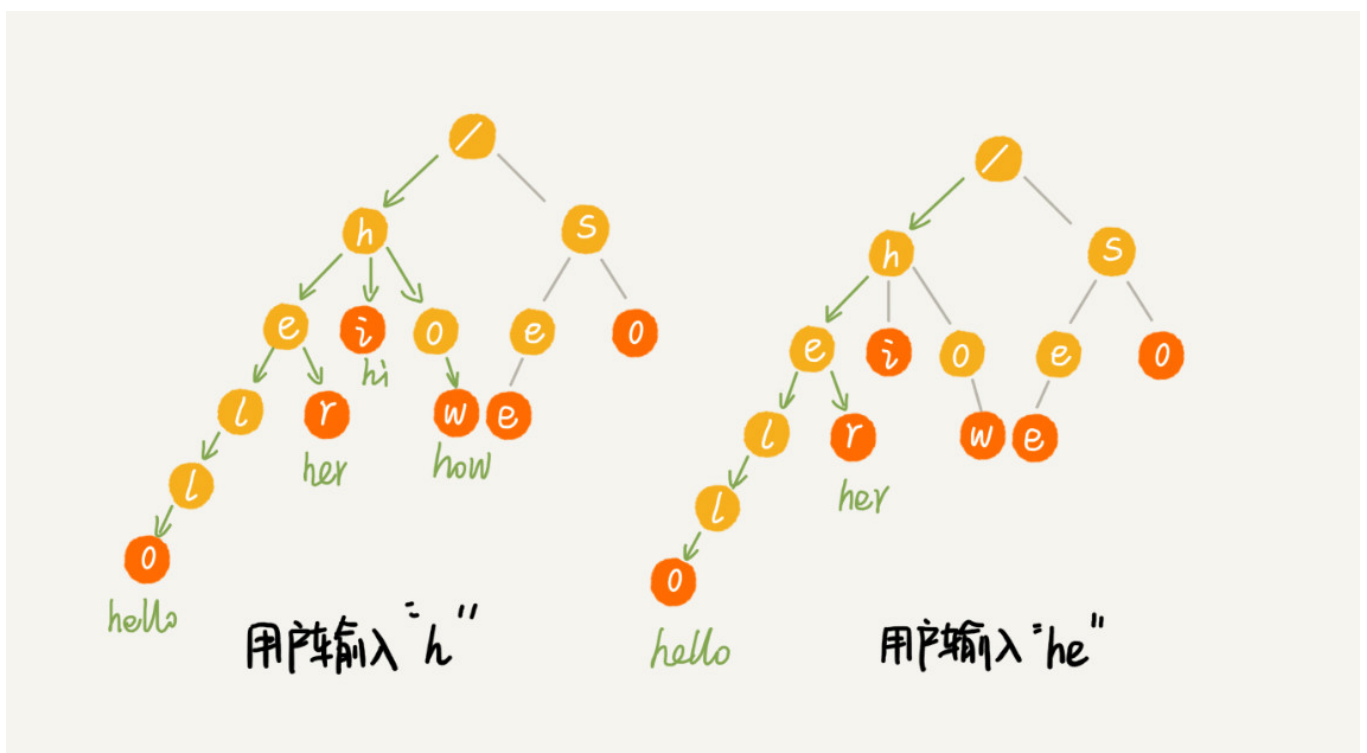
讲到这里，你可能要疑惑了，讲了半天，我对 Trie 树一通否定，还让你用红黑树或者散列表，那 Trie 树是不是就没用了呢？是不是今天的内容就白学了呢？

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。为了讲解方便，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。

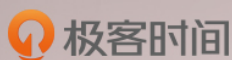
尽管比较耗费内存，但是对内存不敏感或者内存消耗在接受范围内的情况下，在 Trie 树中做字符串匹配还是非常高效的，时间复杂度是  $O(k)$ ， $k$  表示要匹配的字符串的长度。

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



上一篇 34 | 字符串匹配基础（下）：如何借助BM算法轻松理解KMP算法？

下一篇 36 | AC自动机：如何用多模式串匹配实现敏感词过滤功能？

写留言

### 精选留言



ban

11

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？

2018-12-13



kepmov

4

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

2018-12-12

#### 作者回复

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

2018-12-13



Smallfly

3

思考题：

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。

2018-12-12



传说中的成大大

2

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

2018-12-13



夏洛克的救赎

2

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？

2018-12-12



ZX

👍 1

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

2018-12-17

作者回复

那个更高级 不讲了 自学吧亲

2018-12-18



起点·终站

👍 1

看完后发现我们项目的屏蔽字检测就是用trie树写的。。666

2018-12-17



传说中的成大大

👍 1

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？

2018-12-13



等风来

👍 1

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

2018-12-12

作者回复

嗯嗯 怕看不懂嘛

2018-12-12



Lisa Li

👍 0

“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。”可以解释一下为什么吗？

2018-12-18

作者回复

用指针串起来的数据在内存中是不连续的 对cpu缓存来说 不友好 你可以看下链表那一节

2018-12-19



Li Yao

👍 0

p = p.children[index];

p.isEndingChar = true;

是不是应该放到for循环外面？ 否则每个节点都会被标记为endingChar？

2018-12-17

作者回复

已经改正 多谢🙏

2018-12-19



feifei

0

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

- 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:  $\text{count}(\text{value} > 1) / \text{count}(\text{all})$

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

2018-12-16



莫弹弹

0

想起来上一次需求是做敏感词检测，需要匹配到指定动词和指定名词才会触发屏蔽，经过实验，暴力for循环是最快的("⌘⊗") trie太复杂了写完怕别人看不懂



2018-12-14

## 作者回复

这个数据量小的话 确实暴力最快

2018-12-14

我爱学习

刘远通

👍 0

构造一个特殊的hash函数  
可以反映重复度的

比如26进制

然后变成1万个数 看聚点多不多

设置一个 $\epsilon$  两个点之间距离小于 $\epsilon$ 就合并在一起 并且取平均

不断聚合 直到无法再聚合

计算点的个数

2018-12-13



良辰美景

👍 0

1: 上面代码里: `p.isEndingChar = true`; 应该是放在for循环的外面吧?

2: 思考题里的何统计字符串的字符集大小, 这个可以用散列表 (hashmap) key为字符, 值为出现的次数。

3: 前缀统计的话, 其实也可以用hashmap, 只是一个字符串的前缀会有很多, 空间上会浪费些。

4: 如果对字符串按字典排序的话, 在统计前缀上应该会快点, 但我还是没想好用什么数据结构统计。

2018-12-13



邵靖隆

👍 0

王老师, 我今天才知道2-4树和红黑树是等价的, 可以互化, 您可以仔细讲讲这一块吗, 我觉得只要讲通了这一块, 就能同时明白红黑树和B树了

2018-12-13



子嘉

👍 0

咋感觉中文的可以用 有向图? 文字类似于微博的关系 “我们” 是 我->们 但是不是 们->我就是实现起来感觉非常麻烦

2018-12-13



🐶 hfy 🐶

👍 0

老师, 请问如果想实现一个带LRU功能的trie树, 有什么比较高效的算法?

2018-12-13



吴...

👍 0

老师, 麻烦请问一下您请问一下对于一些新的领域, 没有现成的书, 教程你会通过什么方法跟途径或者说平台去体系的学习? 还有我就是想请教一下老师您对AR的看法?

2018-12-12

## 作者回复

没有现成的教程 书。只能自己摸索了。多看看别人分享的文章 自己理个知识大纲 然后再查缺补漏吧

2018-12-13



🐱 您的好友William🐱

👍 0

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！

2018-12-12



『LHCY』

👍 0

字符范围的话用一个HashSet，前缀重合度可以直接构建一个trie树用（10000-总结点个数）/10000。

2018-12-12



narcos

👍 0

老师，下一节是 AC 自动机吗？什么时候发？

2018-12-12

## 作者回复

下一节

2018-12-12



blacknhole

👍 0

（一个疑问

"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)。"中所说的红色节点在文中一直都是叶子节点啊，为什么要提示不都是叶子节点？)

我想到了，红色是isEndingChar为true，但依然可以有子节点，即作为其他字符串的前缀。

2018-12-12



freeland

👍 0

以太坊上header里的transaction .root ,state root,receipt root用的是Merkle-PatriciaTrie(MPT)，和今天的这个是一个么

2018-12-12

## 作者回复

思路差不多 更高级点

2018-12-13



blacknhole

👍 0





一个疑问：

“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）。”中所说的红色节点在文中一直都是叶子节点啊，为什么要提示不都是叶子节点？

2018-12-12



NeverMore

👍 0

思考题我的一点想法：按照Trie 树的添加方式添加完所有字符串，如果树的分支多于某个值（树叉太多了，则表明不匹配的太多），则说明不满足，这个值可以用用户自己设置。

2018-12-12



小美

👍 0

老师 红黑树 如何实现字符串查找 方便王老师稍微点拨下吗

2018-12-12

作者回复

把字符串整体当做一个数据 存储在节点里

2018-12-13



hughieyu

👍 0

```
if (i == text.length-1) {  
    newNode.isEndingChar = true;  
}
```

这一段代码是不是应该拿到上一层判断是否存在的if外面去 比如先走hello 再有he e就不是endchar了

2018-12-12

作者回复

这里是完全匹配 不考虑前缀的情况

2018-12-12