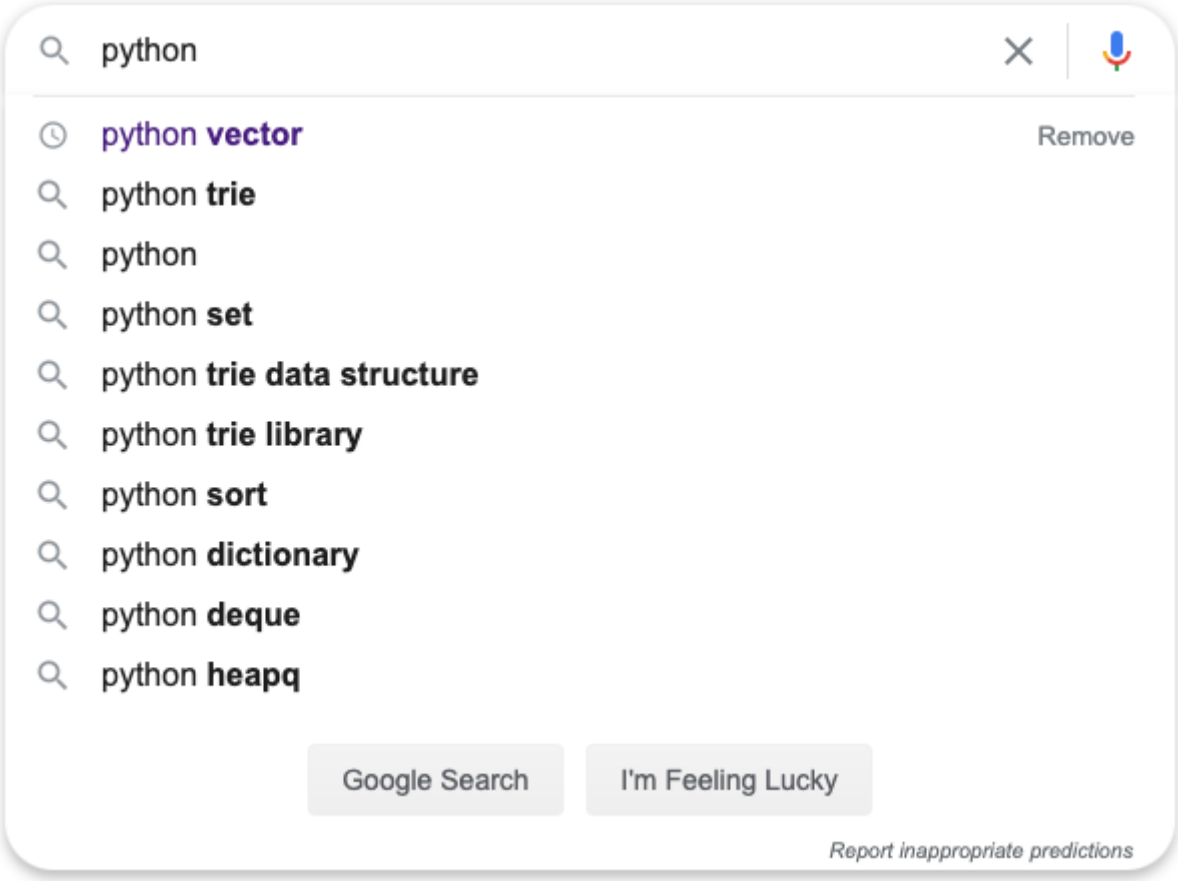
巧用 Trie 树实现搜索引擎关键词提示功能

原创 码海 码海 2020-05-16

收录于话题
#搜索引擎 188 #算法 2539 #Trie树 2

前言

我们几乎每天都在用搜索引擎搜索信息，相信大家肯定有注意过这样一个细节:当输入某个字符的时候，搜索引框底下会出现多个推荐词，如下，输入「python」后，底下会出现挺多以**python 为前缀**的推荐搜索文本，它是如何实现的呢？



文章标题已经给出答案了，没错，用 Trie 树。本文将会从以下几个方面来简述一下 Trie 树的原理，以让大家对 Trie 树有一个比较全面的认识。

- 什么是 Trie 树
- Trie 树的实现

- 如何实现搜索字符串自动提示
- 再谈 Trie 树

相信大家看了肯定有收获

什么是 Trie 树

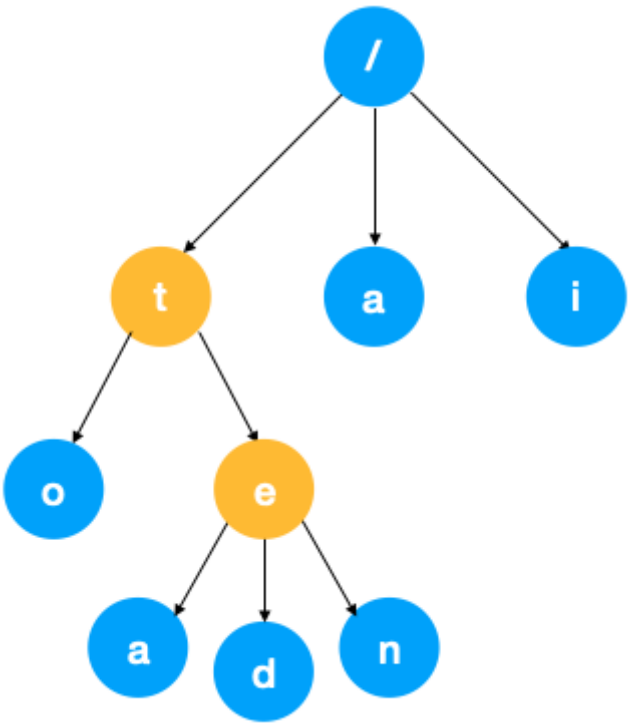
Trie 树，又称前缀树，字典树，或单词查找树，是一种树形结构，也是哈希表的变种，它是一种专门处理字段串匹配的数据结构，用来解决在一组字符串集合中快速查找某个字符串的问题，主要被搜索引擎用来做文本词频的统计。

画重点：快速字符串匹配，词频统计。

1、快速字符串匹配

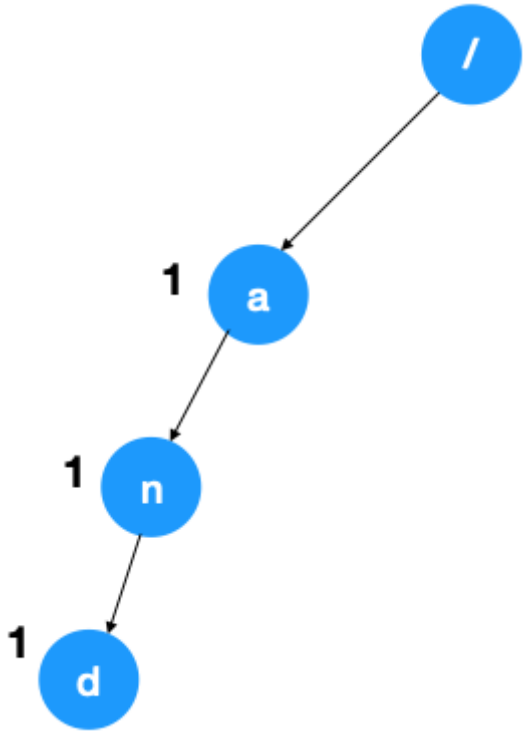
假设想要在一串字符串如 a, to, tea, ted, ten, i, in, inn 中多次查找某个字符串是否存在，该怎么做呢，很直观的想法是用 hash，这种确实没问题，**如果 hash 函数设计得好的话**，如果 hash 函数设计得不好，很容易产生冲突，进而退化成字符串间的比较，另外，在英文中其实有很多单词有共同的前缀，比如中 tea, ted, ten 这三个单词有共同的前缀 te, 如果用 hash 的话，无疑这些共同前缀相当于重复存了多次，比较费空间。

如果用 Trie 树的话，能解决以上两个问题，先来看下 trie 树是如何表示的，以以上的一组字符串 a, to, tea, ted, ten, i, in, inn 为例，它们组成的 Trie 树如下：



如果要查找某个字符串的话，从根节点出发，每次取待查找字符串中的一个字符往下遍历，即可找到，可以看到它的查找时间复杂度为 O(N) (N 为字符串长度)，还是很快的（英文单词普遍比较短）。

2、词频统计 只要在每个结点上加一个计数器，遍历单词时，所有字符串的最后一个字符对应结点的计算器都加 1， 如以 a，an，and 构造的 Trie 树如下，每个结点计算器都为 1，代表以此结点存储字符为终止字符的单词分别为 1 个。



从前面 Trie 树的图解可以看到 Trie 树的本质就是前缀树，通过提取出字符串的公共前缀（如果有的话），以达到快速匹配字符串的目的。

通过前缀匹配，使用 Trie 树查找字符串的效率大大提高！

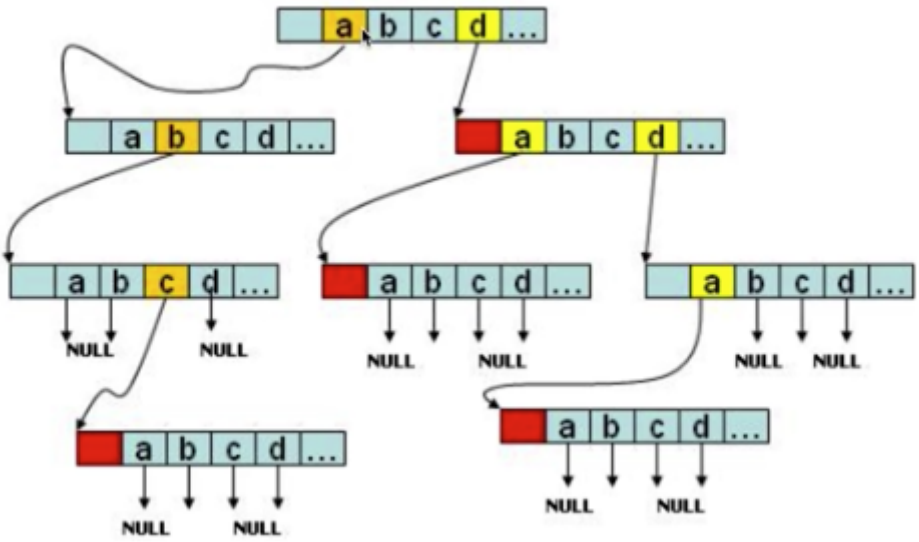
从以上 Trie 树的图解我们可以得出 Trie 树的以下几个特点

1. 根节点不包含字符，除根节点外每个节点只包含一个字符
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。如上图中从根节点到结点 o，经过的字符为「t」和「o」,所以它表示单词 to。
3. 每个节点的所有子节点包含的字符都不相同，这一点也就保证了相同的前缀能够得到复用。

那么 Trie 树该怎么表示呢

Trie 树的实现

从上文我们对 Trie 树的剖析可以很明显地看到 Trie 树是一颗多叉树，那么多叉树该怎么表示呢，假设字符串都是由 26 个小写字母组成，则显然 Trie 树应该是一颗 26 叉树，每个节点包含 26 个子节点，如下



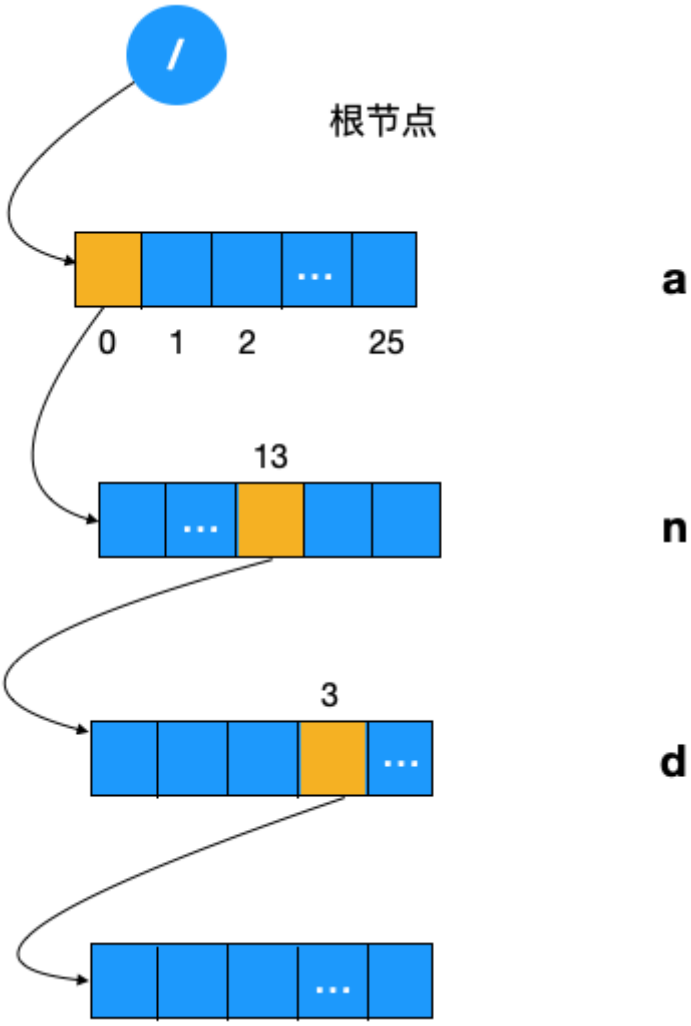
上图可以看出，26 个子节点我们可以用大小为 26 的数组表示，所以 Trie 树表示如下

```
/**
 * 26 个字母
 */
static final int ALPHABET_SIZE = 26;

/**
 * Trie 树的节点表示
 */
static class TriedNode {
    /**
     * 根节点专用，存储 "/"
     */
    public char val;

    /**
     * 以此结点字符为终止字符的字符串的个数
     */
    public int frequency;

    /**
     * 节点指向的子节点
     */
    TriedNode[] children = new TriedNode[ALPHABET_SIZE];

    public TriedNode(char val) {
        this.val = val;
    }
}

/**
 * Trie 树
 */
static class TrieTree {
    private TriedNode root = new TriedNode('/'); // 根节点
}
```

Trie 树的表现有了，现在我们来看下 Trie 树的两个主要操作

1. 根据一组字符串构造 Trie 树
2. 在 Trie 树中查找字符串是否存在

先来看如何根据一组字符串构造 Trie 树，首先如何根据一个单词来构造 Trie 树呢，假设我们以单词 「and」 为例来看下 Trie 树的表现形式



注：图中的数字表示数组的元素位置

可以看到构建 Trie 树的主要步骤如下

1. 构建根节点，此时根节点存有一个元素大小为 26 的数组
2. 遍历字符串「and」
3. 遍历第一个字符 a 时，将上述数组的第一个元素赋值为一个 TriedNode 实例（假设其名为 A）
4. 当遍历第二个字符 n 时，将 A 结点 TriedNode 数组下标为 n-a = 13 (a 的 ascii 为 97，n 的 ascii 码为 110) 的元素赋值为一个 TriedNode 实例（假设其名为 N）
5. 同理，当遍历第三个字符 d 时，将 N 结点 TriedNode 数组的第 4 个元素（下标为 3）赋值为一个 TriedNode 实例（假设其名为 D），同时也将其结点的 frequency 加一，代表以此字符为终止字符的字符串多了一个。

由以上分析不难写出根据字符串构建 Trie 树的代码，如下

```
/**
 * Trie 树
 */
static class TrieTree {
    private TriedNode root = new TriedNode('/'); // 根节点

    /**
     * 以 String 为条件构建 Trie 树
     * @param s
     */
    public void insertString(String s) {
        TriedNode p = root;
        for (int i = 0; i < s.length(); i++){
            char c = s.charAt(i);
            int index  = c-'a';
            if (p.children[index] == null) {
                p.children[index] = new TriedNode(c);
            }
            p = p.children[index];
            //Process char
        }
        p.frequency++;
    }
}
```

Trie 树构造好了，再在 Trie 树中查找某字符串是否存在就简单很多了，遍历字符串，查看每个字符在相应层级的数组位置的元素是否为空即可，如果是，说明不存在，如果不是，则继续遍历字符查找，直到遍历完成，代码如下

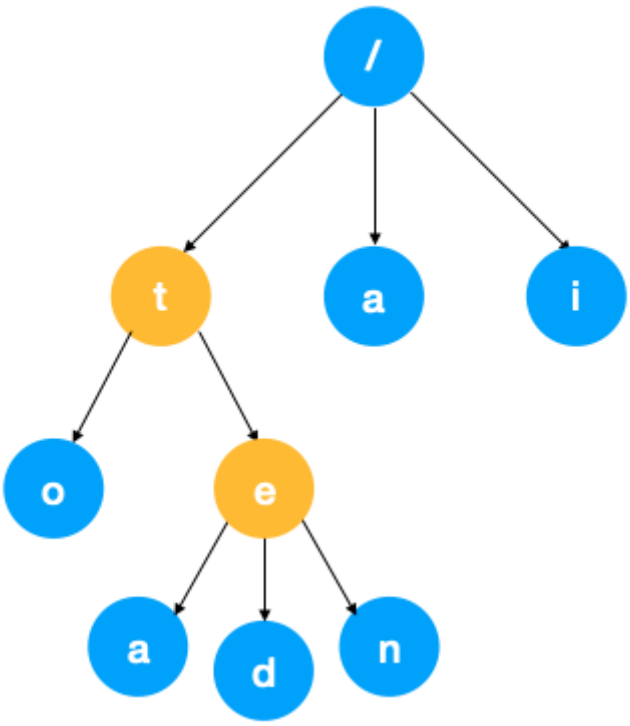
```
/**
 * 查找字符串是否在原字符串集合中
 * @param s
 * @return boolean
 */
public boolean findStr(String s) {
    TriedNode p = root;
    for (int i = 0; i < s.length(); i++){
        // 当前被遍历的字符
```

```
            char c = s.charAt(i);
            int index  = c-'a';
            if (p.children[index] == null) {
                // 如果字符对应位置的数组元素为空，说明肯定不存在此字符，终止之后的字符遍历
                return false;
            }
            // 如果存在，则继续往后遍历字符串
            p = p.children[index];
        }
        return true;
    }
```

由于在节点中也用 frequency 保存了单词数，所以如果在 Trie 树中最终发现字符串存在，也可以随便查找出此字符串的个数。

如何实现搜索字符串自动提示功能

有了 Trie 树，相信大家不难解决开篇的这个问题，首先搜索引擎根据用户的搜索词构建一颗 Trie 树，假设这个搜索词库是 a, to, tea, ted, ten, i, in, inn，则构建的 Trie 树为



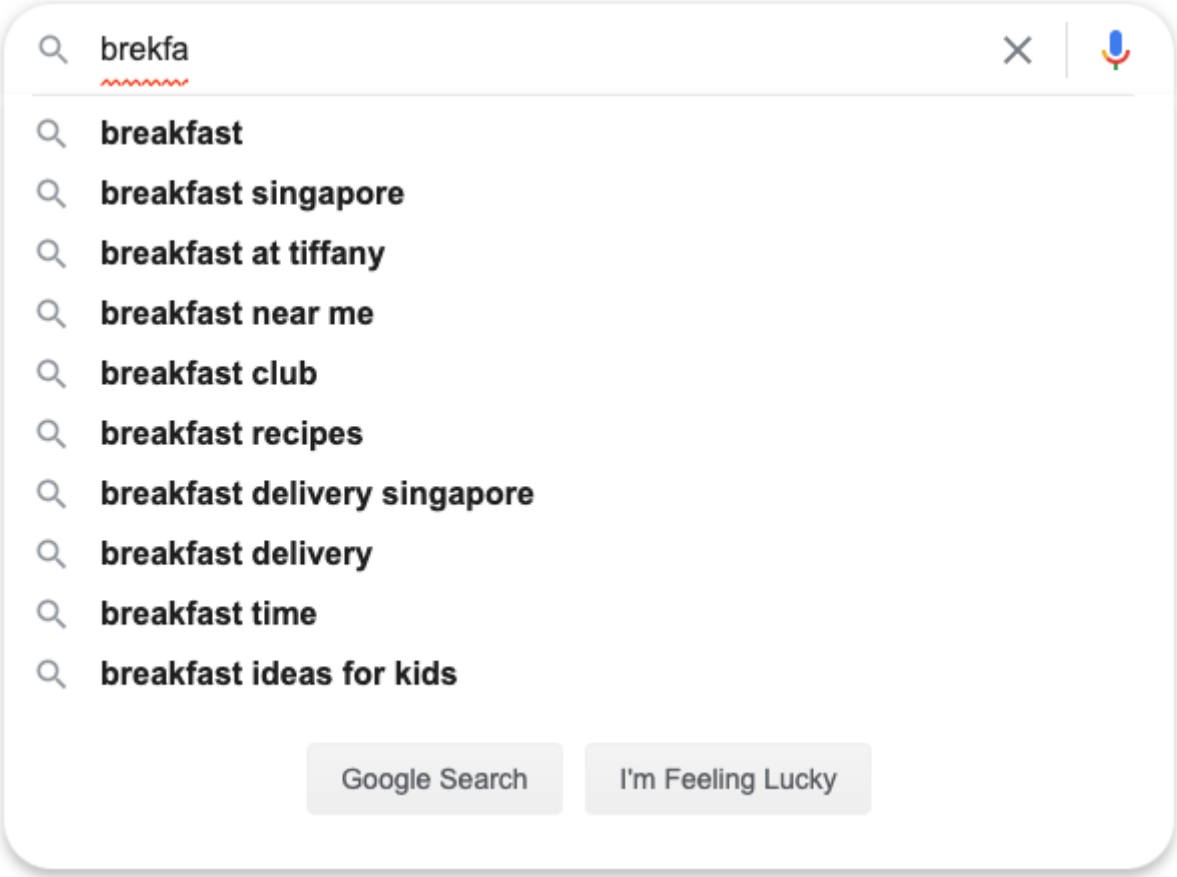
那么当用户在搜索框输入「te」的时候，根据 Trie 树的特性得知以 te 为前缀的字符串有 tea，ted，ten，则应该在搜索框提示词中展示这三个字符串。这里有一个小问题，一般搜索框只会展示 10 个搜索词，但以用户输入字符串为前缀的字符串可能远超 10 次，

到底该展示哪 10 个呢，最简单的规则是展示搜索次数最多的 10 个字符串，于是问题就转化为了 TopK 问题，维护一个有 10 个元素的小顶堆，步骤如下

1. 先根据用户输入的前缀在树中找出含有此前缀的所有字符串
2. 我们知道在节点中保存了字符串的被搜索次数，所以利用小顶堆即可算出被搜索次数最多的 10 个字符串，即可得最终展示给用户的提示词。

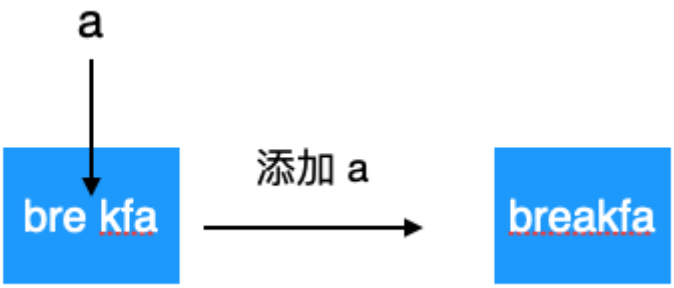
注意：这里的求 TopK 要用是小顶堆，不是大顶堆哦，在搜索引擎背后的经典数据结构和算法这篇文章中有读者提出了疑问，不要搞混了，小顶堆是求最大的 Top K 值，大顶堆是求最小的 TopK 值，由于我们要求最多的前 10 个搜索词，所以应该是用小顶堆）。

这样就解决了，考虑以下现象：我们在输入搜索词的时候，搜索引擎给出的提示词**可能并不是以用户输入的字符串为前缀的**



如图示：搜索引擎给出的搜索关键字并不包含有「brekfa」 前缀。

这种又是怎么实现的呢，它实际上用到了**字符串编辑距离**的思想，所谓字符串编辑距离是说一个字符串可以通过增删改查字符来变成另外一个字符串



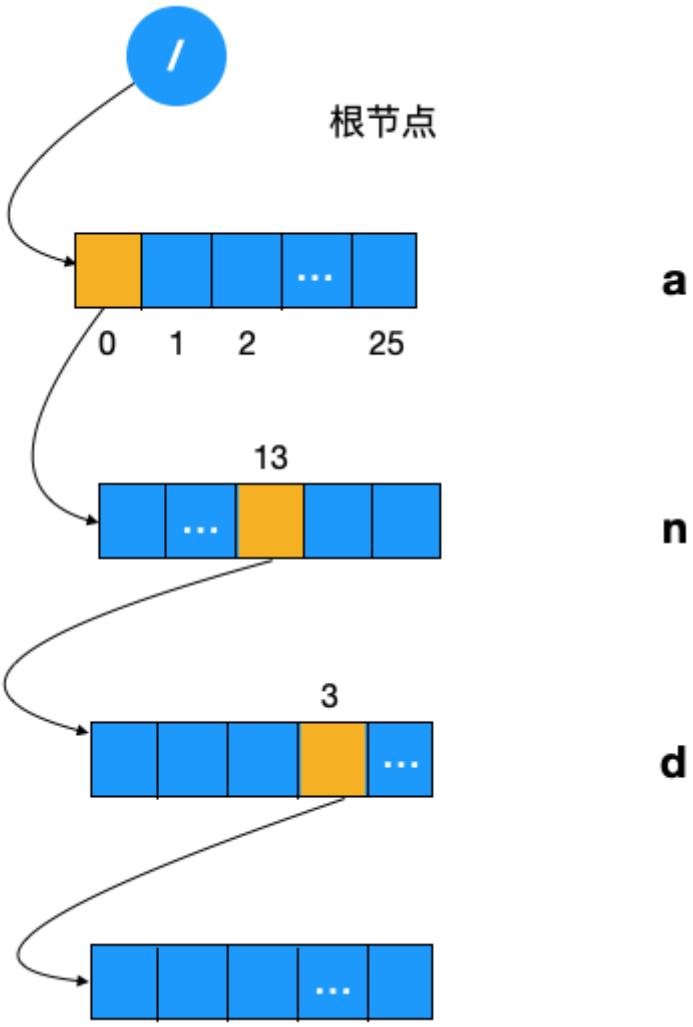
如图示: brekfa 添加 a 之后变成了 breakfa

显然所作的增删改查次数越少，效率越高，经过最少的字符中编辑变成另一个合法的字符串后，就以此字符串为前缀去 Trie 树中查找提示词。

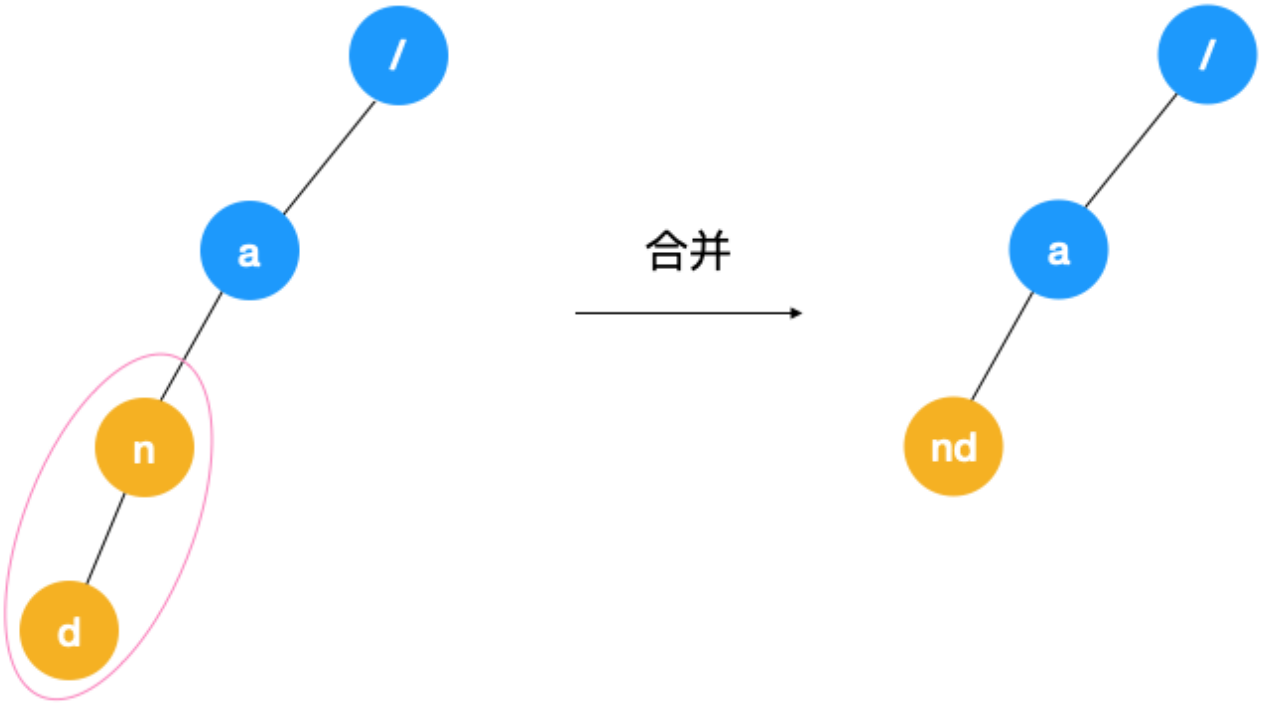
当然了，像 Google 这样的搜索引擎要实时显示这些结果，背后肯定经过了很多改造。不过原理都大同小异。

再谈 Trie 树

从前面的介绍中我们可以看到使用 Trie 树确实在能在快速查找字符串与词频统计上发挥重要作用，但天下没有免费的午餐，如果字符集比较大的话，用 Trie 树可能会造成空间的浪费，以上文中构建的 Trie 树为例



每个结点维护一个 26 个元素大小的数组，共有 4 个数组，也就是分配了 26 x 4 = 104 个元素的空间，但实际上只有三个元素空间（a，n，d）被分配了，浪费了 101 个空间，空间利率率很低，所以一般更适用于字符串前缀重复比较多的情况，当然也可以考虑对 Trie 树进行如下缩点优化，能节省一些空间



当然这么优化后也增加了代码的编码难度，所以要视情况而定。

另外如果用 Trie 树的话，一般需要我们自己编码，对工程师的编码能力要求较高，所以是否用 Trie 树我们一般建议如下：

1. 如果是字符串的精确匹配查找，我们一般建议使用散列表或红黑树来解决，毕竟很多语言的类库都有现成的，不需要自己实现，拿来即用
2. 如果需要进行前缀匹配查找，则用 Trie 树更合适一些

总结

本文通过搜索引擎字符串提示简要地概述了其实现原理，相信大家应该理解了，需要注意的是其使用场景，更推荐在需要前缀匹配查找的时候用 Trie 树，否则像一般的精确匹配查找等更推荐用散列表和红黑树这些很成熟的数据结构，毕竟这两数据结构实现一般在类库中都是实现了的，不需要自己实现，尽量不要重复造轮子。

感谢阅读，欢迎关注公众号一起交流，共同进步！