Trie 树与自动机理论

数据结构翻转课程

王新宇 cp_cp

山东大学泰山学堂计算机取向

青春 Trie 树不会梦到自动机学长

有限状态机（FSM）

- FSM 是给定符号输入，依据转移函数“跳转”过一系列状态的一种机器。

青春 Trie 树不会梦到自动机学长

有限状态机（FSM）

- FSM 是给定符号输入，依据转移函数“跳转”过一系列状态的一种机器。
- 逐个读取输入中的符号，直到被完全耗尽 (把它当作有一个字写在其上的磁带，通过自动机的读磁头来读取它；磁头在磁带上前行移动，一次读一个符号)。一旦输入被耗尽，自动机被称为“停止”了。

青春 Trie 树不会梦到自动机学长

有限状态机（FSM）

- FSM 是给定符号输入，依据转移函数“跳转”过一系列状态的一种机器。
- 逐个读取输入中的符号，直到被完全耗尽 (把它当作有一个字写在其上的磁带，通过自动机的读磁头来读取它；磁头在磁带上前行移动，一次读一个符号)。一旦输入被耗尽，自动机被称为“停止”了。
- 依赖自动机停止时的状态，称呼这个自动机要么是“接受”要么“拒绝”这个输入。如果停止于“接受状态”，则自动机“接受”了这个字。在另一方面，如果它停止于“拒绝状态”，则这个字被“拒绝”。自动机接受的所有字的集合被称为“这个自动机接受的语言”。

一个简单的自动机

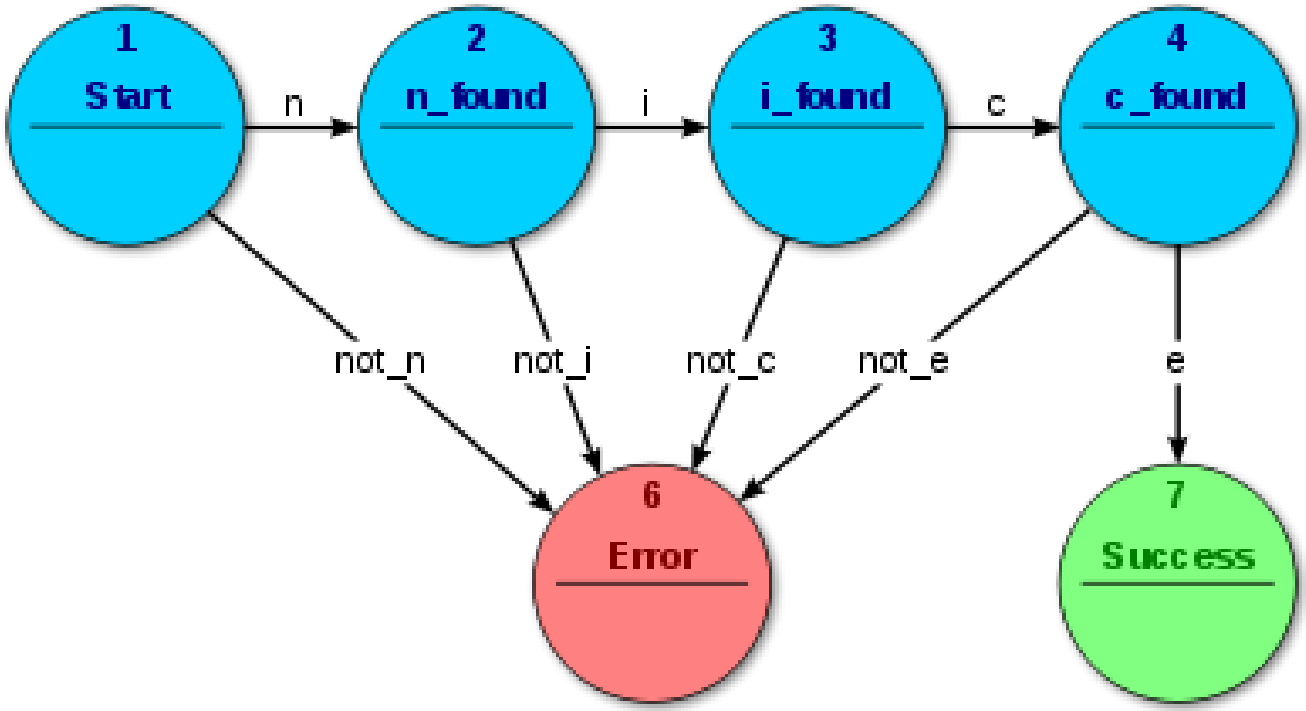


图 1: 一个判断输入是否为“nice”的自动机

不想成为 LaTeX 的自动机不是好的 MarkDown

自动机可以表示为五元组

$$M(Q, \Sigma, \delta, q_0, F)$$

Q 是状态的集合；
Σ 是符号的有限集合，也就是自动机可接受的语言的字母表。
δ 是转移函数：

$$\delta : Q \times \Sigma \to Q$$

q_0 是开始状态，未处理输入时的状态，有 $q_0 \in Q$
F 叫做终止状态，有 $F \subseteq Q$

从状态图到形式化表达

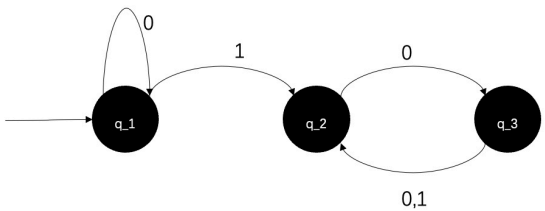


图 2: M_1 的状态图

M_1 形式化写为

1. 状态集合 $Q = \{q_1, q_2, q_3\}$
2. 字母表 $\Sigma = \{0, 1\}$
3. 转移函数 δ 描述为：

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

4. q_1 是初始状态
5. $F = \{q_2\}$

如何设计一个自动机

方法论

1. 确定状态集合 Q，初始状态 q_0，接收状态状态集 F。

如何设计一个自动机

方法论

1. 确定状态集合 Q，初始状态 q_0，接收状态状态集 F。
2. 确定状态转移，包括转移函数 δ，字母表 Σ

如何设计一个自动机

方法论

1. 确定状态集合 Q，初始状态 q_0，接收状态状态集 F。
2. 确定状态转移，包括转移函数 δ，字母表 Σ
3. 画个图（非形式化定义）

一个简单题

制作一个判断输入（二进制字符串）是否为 3 的倍数的自动机

你肯定会，试一试。

简单题的答案

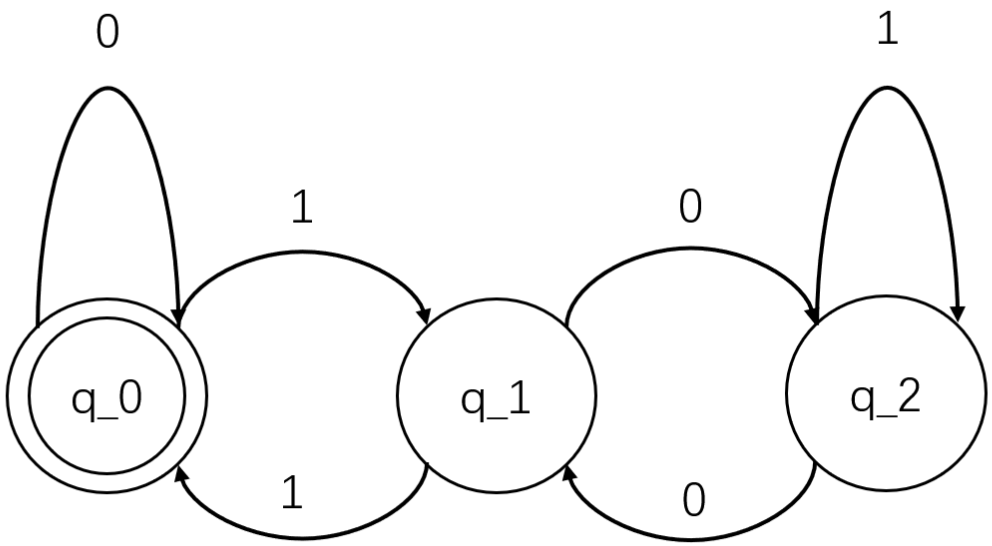


图 3: 简单题的状态图

改进一点点小细节

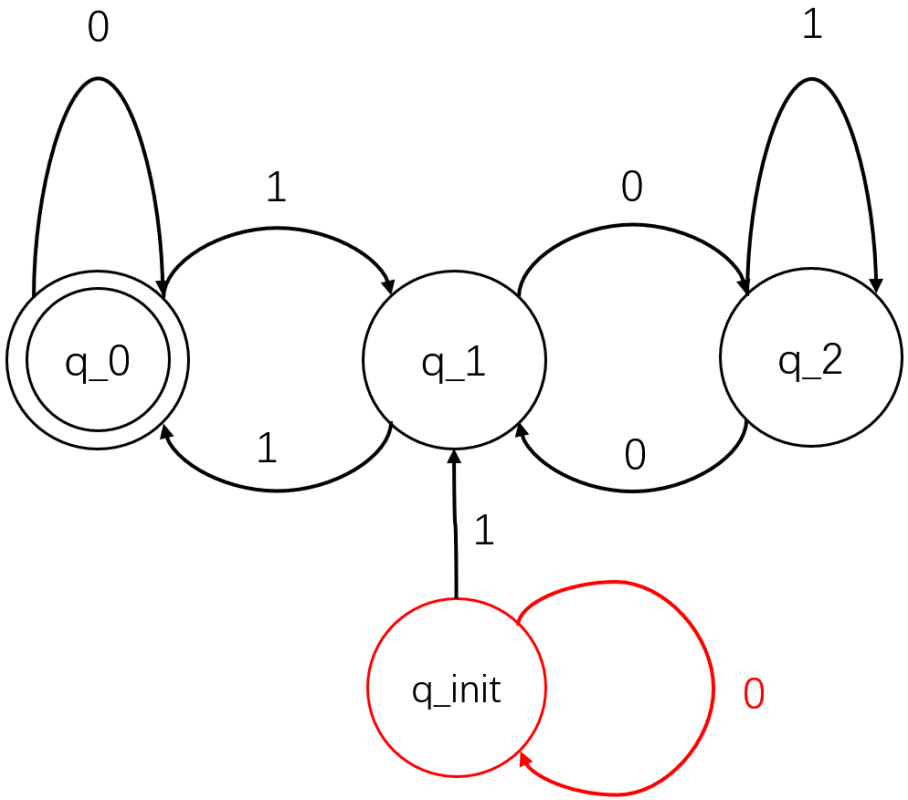


图 4: 修改版简单题

一个单题

制作一个判断输入（十进制字符串）是否为 3 的倍数的自动机

你肯定也会，但会比较耗时间。

一个单题的图

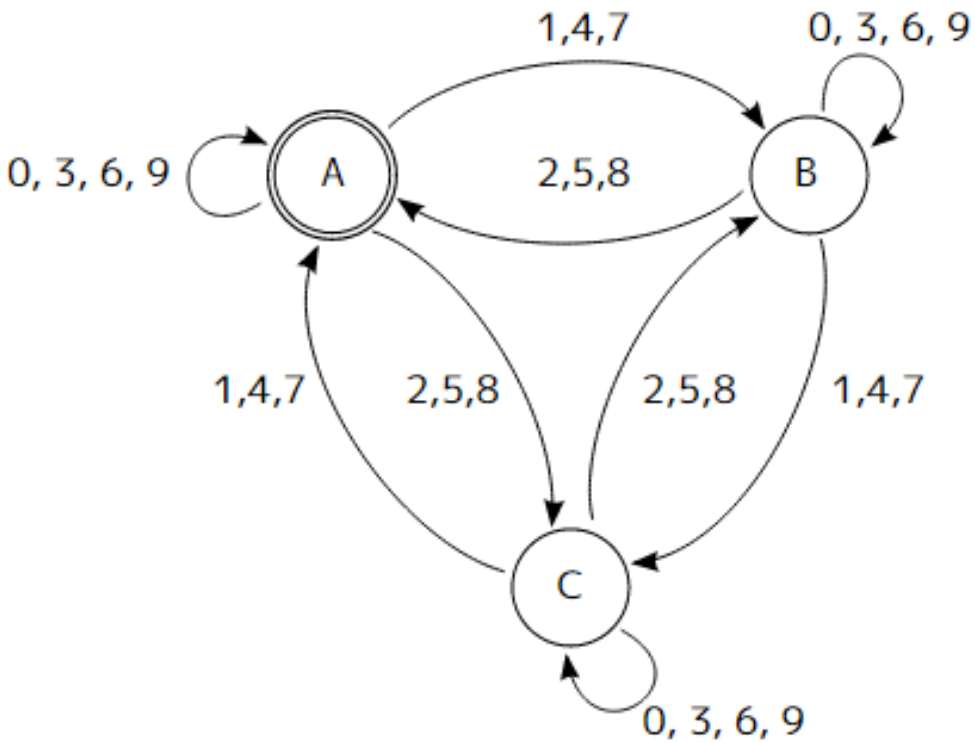


图 5: 有点好看

对应的正则表达式

```
1  ^[0369]* (
2    (
3       [147][0369]*
4    | [258][0369]*[258][0369]*
5    ) ([147][0369]*[258][0369]*)* (
6       [258][0369]*
7
8    | [147][0369]*[147][0369]*
9    )
10 | [258][0369]*[147][0369]* )* $
```

正则表达式如何匹配 3 的倍数？

一些定义

字符串

输入到自动机的字符序列。

字符串与语言的关系

若 A 是机器 M 接受的全部字符串集，则称 A 是机器 M 的语言，记作 $L(M) = A$。

进一步的

非形式化描述易于前期掌握，而形式化定义可以把概念完全搞清楚。
形式化定义也能够清除非形式化描述的二义性。

计算的形式化表达

设 $M = (Q, \Sigma, \delta, q_0, F)$ 是一台有限状态自动机，
$w = w_1 w_2 w_3 ... w_n$ 是一个字符串并且任意一个 w_i 是字母表 Σ 的成员，如果 Q 存在状态序列 $r_0, r_1, ..., r_n$，满足以下条件：

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}, i = 0, ..., n-1$
3. $r_n \in F$

则称 M 接受字符串 w。
如果 $A = \{w | M接受w\}$，则称 M 识别语言 w；
如果一个语言能被一台有限自动机识别，则称它是正则语言。

不好意思，前置有些多

让我们进入正题

名字由来

- 1912 年，Axel Thue 首次抽象地描述了表示一组字符串的 trie 的想法。

名字由来

- 1912 年，Axel Thue 首次抽象地描述了表示一组字符串的 trie 的想法。
- 1959 年，René de la Briandais 首次在计算机环境中描述了尝试。

名字由来

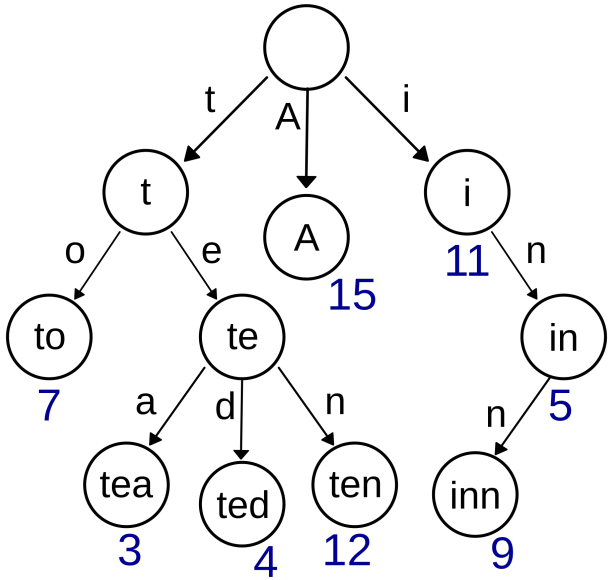
- 1912 年，Axel Thue 首次抽象地描述了表示一组字符串的 trie 的想法。
- 1959 年，René de la Briandais 首次在计算机环境中描述了尝试。
- 1960 年，爱德华·弗雷德金 (Edward Fredkin) 独立地描述了这个想法，他创造了“trie”一词，取检索 **retrieval** 的中间音节，将其发音为/tri/（作为“树”）。

名字由来

- 1912 年，Axel Thue 首次抽象地描述了表示一组字符串的 trie 的想法。
- 1959 年，René de la Briandais 首次在计算机环境中描述了尝试。
- 1960 年，爱德华·弗雷德金 (Edward Fredkin) 独立地描述了这个想法，他创造了“trie”一词，取检索 **retrieval** 的中间音节，将其发音为/tri/（作为“树”）。
- 然而，其他作者将其发音为/tra/（作为“尝试”），试图在口头上将其与“树”区分开来。

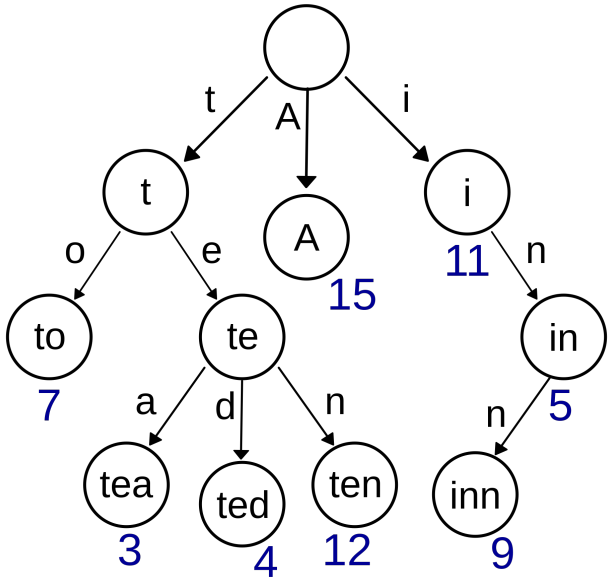
基本概念

- trie，又称前缀树或字典树。



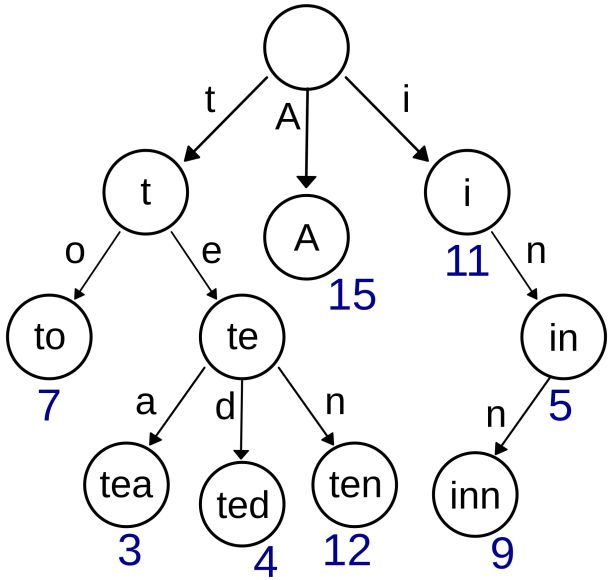
基本概念

- trie，又称前缀树或字典树。
- 是一种有序树，用于保存关联数组，其中的键通常是字符串，值是字符串对应的值。



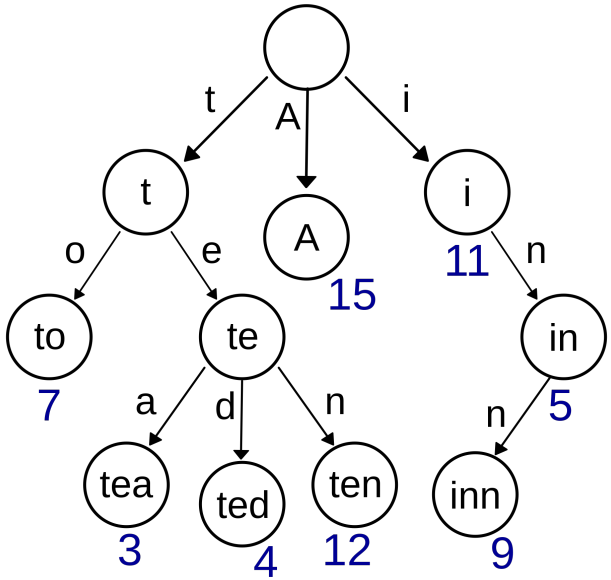
基本概念

- trie，又称前缀树或字典树。
- 是一种有序树，用于保存关联数组，其中的键通常是字符串，值是字符串对应的值。
- 与二叉查找树不同，键不是直接保存在节点中，而是由节点在树中的位置决定。



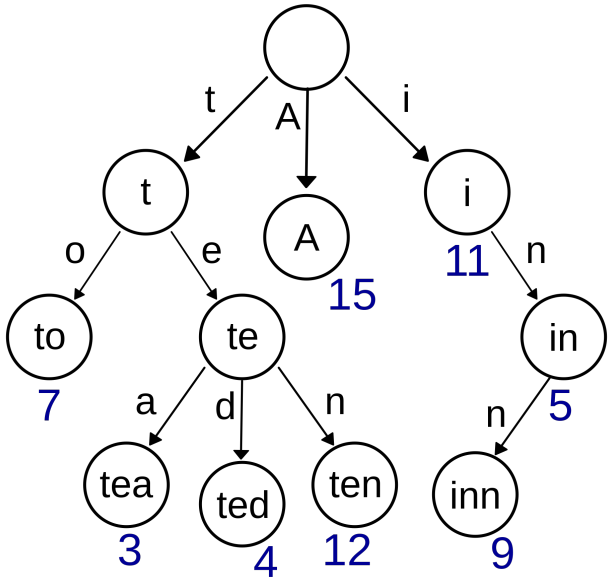
基本概念

- trie，又称前缀树或字典树。
- 是一种有序树，用于保存关联数组，其中的键通常是字符串，值是字符串对应的值。
- 与二叉查找树不同，键不是直接保存在节点中，而是由节点在树中的位置决定。
- 一个节点的所有子孙都有相同的前缀，也就是这个节点对应的字符串，而根节点对应空字符串。



基本概念

- trie，又称前缀树或字典树。
- 是一种有序树，用于保存关联数组，其中的键通常是字符串，值是字符串对应的值。
- 与二叉查找树不同，键不是直接保存在节点中，而是由节点在树中的位置决定。
- 一个节点的所有子孙都有相同的前缀，也就是这个节点对应的字符串，而根节点对应空字符串。
- 一般情况下，不是所有的节点都有对应的值，只有叶子节点和部分内部节点所对应的键才有相关的值。



对于一个 Trie 树

- 每个节点对应一个值（也称作“键值”），我们把节点的键值看作一种状态。

对于一个 Trie 树

- 每个节点对应一个值（也称作“键值”），我们把节点的键值看作一种状态。
- 将对应的键值全体可以看作一个状态集合，设为 Q

对于一个 Trie 树

- 每个节点对应一个值（也称作“键值”），我们把节点的键值看作一种状态。
- 将对应的键值全体可以看作一个状态集合，设为 Q
- 对于 Tire 树可处理的输入字符，将它们称为符号，其全体称为字母表，设为 Σ。

对于一个 Trie 树

- 每个节点对应一个值（也称作“键值”），我们把节点的键值看作一种状态。
- 将对应的键值全体可以看作一个状态集合，设为 Q
- 对于 Tire 树可处理的输入字符，将它们称为符号，其全体称为字母表，设为 Σ。
- 考虑父节点到子节点的过程，是根据读入符号的转移，设这些转移函数为 $\delta : Q \times \Sigma \to Q$

对于一个 Trie 树

- 每个节点对应一个值（也称作“键值”），我们把节点的键值看作一种状态。
- 将对应的键值全体可以看作一个状态集合，设为 Q
- 对于 Tire 树可处理的输入字符，将它们称为符号，其全体称为字母表，设为 Σ。
- 考虑父节点到子节点的过程，是根据读入符号的转移，设这些转移函数为 $\delta : Q \times \Sigma \to Q$
- Trie 树最初处于未处理状态，也就是根节点所对应的状态，称作初始状态，设为 q_0。

对于一个 Trie 树

- 每个节点对应一个值（也称作“键值”），我们把节点的键值看作一种状态。
- 将对应的键值全体可以看作一个状态集合，设为 Q
- 对于 Tire 树可处理的输入字符，将它们称为符号，其全体称为字母表，设为 Σ。
- 考虑父节点到子节点的过程，是根据读入符号的转移，设这些转移函数为 $\delta : Q \times \Sigma \to Q$
- Trie 树最初处于未处理状态，也就是根节点所对应的状态，称作初始状态，设为 q_0。
- 读完一个输入，Trie 所停留位置所对应的状态就是终止状态，终止状态同样对应于一个集合，设为集合 F，并且有 $F \subseteq Q$。

与自动机关系

前置知识有用了

可以发现，Trie 树对应于一个有限确定状态自动机。
之前铺垫那么多终于有点用了。

基础实现

状态实现
转移实现

时间复杂度

- 假设所有字符串长度之和为 n，构建字典树的时间复杂度为 $O(n)$
- 假设要查找的字符串长度为 k，查找的时间复杂度为 $O(k)$

吃吃吃吃吃吃吃吃吃吃

椰 ser 是一个吃货，喜欢吃水果。比如：

- bananananananana

吃吃吃吃吃吃吃吃吃吃

椰 ser 是一个吃货，喜欢吃水果。比如：

- bananananananana
- bananananananb

吃吃吃吃吃吃吃吃吃吃

椰 ser 是一个吃货，喜欢吃水果。比如：

- bananananananana
- banananananananb
- banananananananc

吃吃吃吃吃吃吃吃吃吃

椰 ser 是一个吃货，喜欢吃水果。比如：

- bananananananana
- bananananananb
- bananananananc
- bananananananand

吃吃吃吃吃吃吃吃吃吃

椰 ser 是一个吃货，喜欢吃水果。比如：

- bananananananana
- bananananananb
- bananananananc
- bananananananand
- bananananananane

吃吃吃吃吃吃吃吃吃吃

椰 ser 是一个吃货，喜欢吃水果。比如：

- bananananananana
- banananananananb
- banananananananc
- banananananananand
- bananananananane
- ……

吃吃吃吃吃吃吃吃吃吃

椰 ser 是一个吃货，喜欢吃水果。比如：

- bananananananana
- banananananananb
- banananananananc
- bananananananand
- bananananananane
- ……
- banananananananz

吃吃吃吃吃吃吃吃吃吃

椰 ser 是一个吃货，喜欢吃水果。比如：

- bananananananana
- banananananananb
- banananananananc
- bananananananand
- bananananananane
- ……
- banananananananz
- 但椰 ser 要把水果存到 Trie 树，这可太长了。

吃吃吃吃吃吃吃吃吃吃

椰 ser 是一个吃货，喜欢吃水果。比如：

- bananananananana
- banananananananb
- banananananananc
- bananananananand
- bananananananane
- ……
- banananananananz
- 但椰 ser 要把水果存到 Trie 树，这可太长了。
- 但是观察一下，可以把中间重复的单元进行压缩。

Redix Tree

这就引出了基数树。

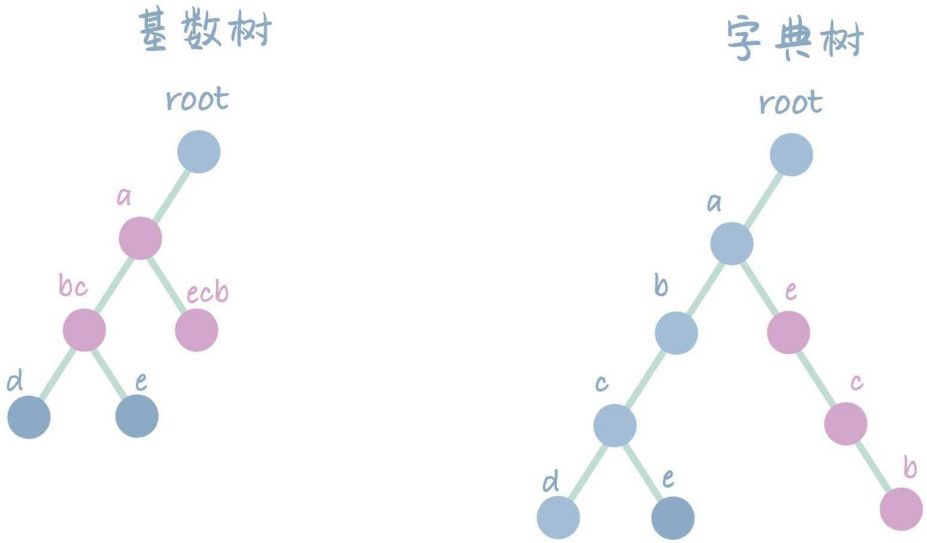


图 6: 基数树

优点

Trie 树的核心思想是空间换时间

- 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。

优点

Trie 树的核心思想是空间换时间

- 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。
- Trie 树中不同的关键字不会产生冲突。

优点

Trie 树的核心思想是空间换时间

- 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。
- Trie 树中不同的关键字不会产生冲突。
- Trie 树只有在允许一个关键字关联多个值的情况下才有类似 hash 碰撞发生。

优点

Trie 树的核心思想是空间换时间

- 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。
- Trie 树中不同的关键字不会产生冲突。
- Trie 树只有在允许一个关键字关联多个值的情况下才有类似 hash 碰撞发生。
- Trie 树不用求 hash 值，对短字符串有更快的速度。通常，求 hash 值也是需要遍历字符串的。

优点

Trie 树的核心思想是空间换时间

- 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。
- Trie 树中不同的关键字不会产生冲突。
- Trie 树只有在允许一个关键字关联多个值的情况下才有类似 hash 碰撞发生。
- Trie 树不用求 hash 值，对短字符串有更快的速度。通常，求 hash 值也是需要遍历字符串的。
- Trie 树可以对关键字按字典序排序。

缺点

- 字符串的字符集不能过大，否则存储空间过于浪费，即便是采用优化方案，也是在牺牲部分查询性能的基础上的

缺点

- 字符串的字符集不能过大，否则存储空间过于浪费，即便是采用优化方案，也是在牺牲部分查询性能的基础上的
- 在字符串前缀重合较多的情况，才有比较好的性能表现

缺点

- 字符串的字符集不能过大，否则存储空间过于浪费，即便是采用优化方案，也是在牺牲部分查询性能的基础上的
- 在字符串前缀重合较多的情况，才有比较好的性能表现
- 字典树中使用到了指针，因此前后节点是不连续的，对 CPU 缓存不友好

缺点

- 字符串的字符集不能过大，否则存储空间过于浪费，即便是采用优化方案，也是在牺牲部分查询性能的基础上的
- 在字符串前缀重合较多的情况，才有比较好的性能表现
- 字典树中使用到了指针，因此前后节点是不连续的，对 CPU 缓存不友好
- 当 hash 函数很好时，Trie 树的查找效率会低于哈希搜索。

一些应用

- 字符串检索:
  - 检索/查询功能是 Trie 树最原始的功能。
- 字符串排序
  - Trie 树可以对大量字符串按字典序进行排序.
- 前缀匹配
  - trie 树前缀匹配常用于搜索提示。
- IP 路由表
  - 在 IP 路由表中进行路由匹配时，要按照最长匹配前缀的原则进行匹配。
- 拼写检查
  - 例如，在 word 中输入一个拼写错误的单词，它能够自动检测出来。
- 辅助结构
  - 作为其他数据结构和算法的辅助结构如后缀树，AC 自动机等。

回顾 KMP 算法的内容

- 给定一个文本 t 和一个字符串 s，尝试找到并展示 s 在 t 中的所有出现。
- 文本 t 称为输入串。
- 字符串 s 称为模式串。

建立 KMP 自动机

对于模式串

我们通过 Trie 树的形式建立，形成一串前缀状态序列。
例如，假设模式串是"tomato"

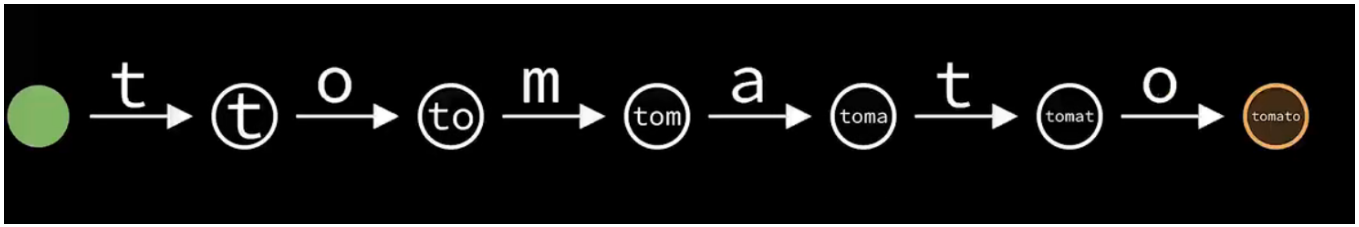


图 7: 模式串的 Trie 树

建立 KMP 自动机

对于输入串

对于每一个位置，以当前位置作为字符串结尾，单独形成一个字符串，通过 Trie 树进行后缀匹配。对于上面例子，Trie 树可以识别出：

- ...t
- ...to
- ...tom
- ...toma
- ...tomat
- ...tomato

按道理来讲，我们只需要以每一个位置当作结尾，就可以找出输入串的所有匹配字符串了。

观察一下

- 特殊的，对于一个字符串来说，在 Tire 树中可能有多个匹配的状态。
- 如“atomat”，可以与"..t","..tomat" 等多个状态进行匹配。可以发现，但是较短的后缀是较长的后缀的后缀。

建立 KMP 自动机

梳理一下关系

- 我们说，深度是节点到根节点的距离。
- 深度深的节点可以通过深度浅的节点转移而来。
- KMP 自动机的状态：最长的等于模式串某个前缀的输入串的后缀。

考虑失配

- 深度浅的节点的信息包含了深度深的节点信息（深度深的节点比深度浅的节点更加严格）
- 那么当在深度深的节点失配时，应当退回最近的深度较浅的（限制更少的）节点，我们称为失配链接。
- 当前节点在上一个节点的失配链接中判断下一个是否匹配。

AC 自动机能不能自动 AC?

AC 自动机大家可能没听说过，但自动 AC 机大家应该了解吧。就如同 KMP 不是“看猫片”一样，但 AC 自动机的 AC 也不是 Accepted，而是是人名 Aho–Corasick

图 8: 什么是猫片

AC 自动机

AC 自动机是一种高效的多模式匹配算法。它可以在一个文本串中同时查找多个模式串。

与 KMP 的区别，在于模式串有多个。但是此时，Trie 更能发挥它的作用，从序列形式，形成一个真正的树形。

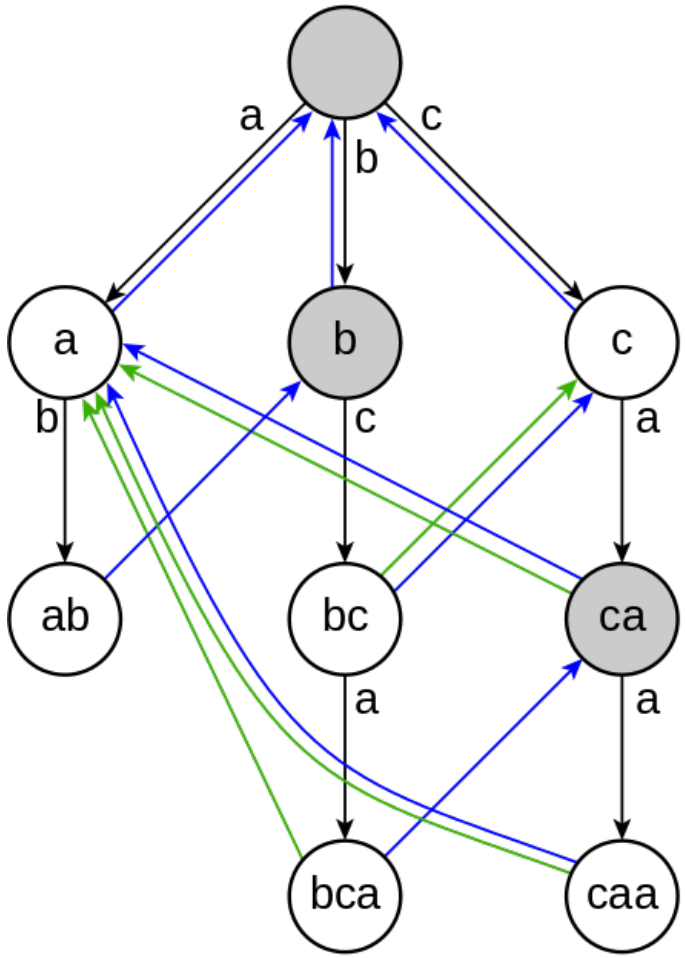


图 9: AC 自动机

[1] Wikipedia contributors. (2023, November 6). Trie. Wikipedia. https://en.wikipedia.org/wiki/Trie

[2] 前缀函数与 KMP 算法 - OI Wiki. https://oi-wiki.org/string/kmp

[3] AC 自动机 - OI Wiki. https://oi-wiki.org/string/ac-automaton/

[4] AC 自动机 - ACWing. (n.d.). https://www.acwing.com/blog/content/34898/

[5] Aho–Corasick algorithm. Wikipedia. https://en.wikipedia.org/wiki/Aho

[6] 千嶂夹城. (n.d.-b). [Video]. https://www.bilibili.com/video/BV1uV4y1W7cB

[7] Bo L. (2019, July 1). 数据结构与算法: 字典树 (Trie). Aimuke. https://aimuke.github.io/algorithm/2019/07/01/algorithm-trie/

Thanks!