

后缀自动机 (姬)

算法思想

后缀自动机简称 SAM

我们记 Σ 为字符集 $|\Sigma|$ 为字符集大小。

以下问题可以通过 SAM 在线性时间内解决。

1. 在另一个字符串中搜索一个字符串的所有出现位置 (KMP 哈希也可以)
2. 计算给定的字符串中有多少个不同的子串。 (后缀数组也可以线性做)

在直观上, 我们可以把 SAM 理解为将字符串的所有子串压缩在一个树上。对于长度为 n 的字符串 SAM 的空间复杂度是 $O(n)$ 的, 此外构造 SAM 的时间复杂度也是 $O(n)$ 的, 小结论: 一个 SAM 最多有 $2n-1$ 个结点和 $3n-4$ 条转移边。

我们把结点称作状态, 边称为状态之间的转移。我们有一个初始的源点作为初始状态, 其他各个结点都可以从这个源点出发到达。每个转移都标有一些字母, 从一个结点出发的所有转移都不同。存在一个或多个终止状态, 路径上所有转移连接起来一定是字符串的一个后缀, 并且每一个后缀都可以用一条从源点到终止状态的路径构成。所以子串就是后缀的前缀也就是从源点开始到任意一个点的路径, 所以可以说一个点对应着一个原字符串的子串。

结束位置 $endpos$ 考虑字符串 s 的任意非空子串 t 我们记 $endpos(t)$ 为在字符串 s 中 t 的所有结束位置 (从 0 开始), 比如现在有一个字符串叫 $abcbcb$ 我们有 $endpos("bc")=2,4$ 不同子串 t_1 和 t_2 的 $endpos$ 集合可能相等, 比如刚才的 cb 和 bcb 所以我们可以根据 $endpos$ 集合的不同将 s 的非空子串分为若干等价类。

SAM 中的每个状态对应一个 $endpos$ 相同的等价类, 还有一个初始状态, 所以 SAM 的状态个数等于等价类的个数 $+1$ 。

假设字符串 s 的两个非空子串 u 和 v 的 $endpos$ 相同, 显然有短的字符串是长的字符串的后缀 $endpos$ 之间要么相交是空 (不为后缀关系), 要么是被包含的关系 (其中一个是另一个的后缀), 且满足这个等价类的子串的长度是一个连续的区间。

后缀链接 $link(v)$ 连接到对应于比 v 等价类最短的还短一个字符的字符串的等价类。所有的后缀链接构成一棵根节点为 t_0 的树。如果我们从任意状态 v_{i_0} 开始沿着后缀链接遍历, 总会到达初始状态 t_0 这种情况会得到一个互不相交的区间 $[\min(len(v_{i_1}), len(v_{i_2})), len(v_{i_0})]$ 并且他们的交集正好是 $[0, len(v_{i_0})]$

后缀自动机的构造是在线的, 我们可以逐个加入字符串的每个字符, 并且每一步维护 SAM 我们为了保证线性的空间复杂度, 只保存 len 和 $link$ 的值和每个状态的转移列表, 不会标记终止状态。一开始 SAM 只包含一个状态 t_0 编号为 0 , 为了方便我们规定 t_0 的 $len=0, link=-1$ (-1 表示虚拟状态)。现在我们开始插入字符: 令 $last$ 为添加字符 c 之前, 整个字符串对应的状态 (每次的最后一步都会更新这个值)。创建一个新的状态 cur (因为整个字符串一定是第一次出现的), 并将 $len(cur)$ 赋值为 $len(last)+1$ 这时 $link(cur)$ 的值还未知。

从状态 $last$ 开始, 如果没有字符 c 的转移, 我们就添加一个到状态 cur 的转移, 遍历后缀链接, 如果在某个点已经存在到字符 c 的转移, 我们就停下来, 并将这个状态标记为 p

如果没有找到这个 p 则到达了 -1 , 此时我们将 $link(cur)$ 赋值为 0 并退出。

如果找到了，且我们知道经过字符 c 转移后的状态为 q 现在分情况进行讨论：

如果 $\text{len}(p)+1=\text{len}(q)$ 我们只需要将 $\text{link}(\text{cur})$ 赋值为 q 并退出。

否则我们需要复制 q 我们创建一个新的状态 clone 复制 q 除了 len 的值之外的所有信息（包括后缀链接和转移），然后将 $\text{len}(\text{clone})$ 赋值为 $\text{len}(p)+1$ 之后我们将 $\text{link}(\text{cur})$ 指向 clone 也将 $\text{link}(q)$ 指向 clone 这样就是 q 发现了一个介于原来两个之间的状态，于是需要把 $\text{link}(q)$ 指向 clone

无论哪种情况，最后我们都将 last 的值更新为 cur

这里遍历后缀链接的原因是：我们创建了一个新的状态之后，对后缀链接这些状态进行遍历，尝试添加通过一个字符 c 到新状态 cur 的转移，但是我们不能覆盖之前的合法转移，当没有找到时，说明这个字符从未出现过，所以后缀链接为 0 。如果出现了，说明我们正在向自动机内添加一个已经存在了的子串，如果存在 $\text{len}(q)=\text{len}(p)+1$ 说明之前已经有一个状态的转移是一样的，直接连到转移后的就可以了；如果不存在，说明转移是不连续的，即 q 不仅对应于长度为 $\text{len}(p)+1$ 的后缀，还对应着更长的子串，就需要拆开状态 q 来创建这样的状态。但是这样还没完，我们需要把一些本来转移到 q 的转移重定向到 clone 我们需要继续沿着后缀链接遍历，从结点 p 直到 -1 或者转移到不是状态 q 的一个转移。

因为我们只为 s 的每个字符创建了一个或者两个新状态，所以 SAM 只包含线性个状态。

算法实现

如果你用 map 存储转移列表，时间复杂度会变成 $O(n \log |\text{sum}|)$ 档子腹肌为较小的常数，比如 26 时，就将转移数组设为 $\text{int}[26]$ 即可。

给 SAM 赋予树形结构，树的根为 0 ，其余结点 v 的父亲为 $\text{link}(v)$ 则 $S_{\{1\dots p\}}$ 和 $S_{\{1\dots q\}}$ 的最长公共后缀对应的字符串就是 $v_{\{p\}}$ 和 $v_{\{q\}}$ 对应的 LCA 的字符串。显然每个状态对应的子串种类数是 $\text{len}(i)-\text{len}(\text{link}(i))$

From: <https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link: https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:%E7%8E%8B%E6%99%BA%E5%BD%AA:%E5%90%8E%E7%BC%80%E8%87%AA%E5%8A%A8%E6%9C%BA&rev=1627489631

Last update: 2021/07/29 00:27