后缀自动机

HZL

说明

- 是按照SAM论文的顺序来讲的
- 论文中下标从0开始，不过一般做题还是从1开始，不影响中间的理解
- 俄语原文：https://e-maxx.ru/algo/suffix_automata
- 英语译文：https://cp-algorithms.com/string/suffix-automaton.html#suffix-links-link
- Oi-wiki：https://oi-wiki.org/string/sam/

自动机

- OI中的“自动机”一般说的是“确定有限状态自动机（DFA）”
- 一个DFA由以下五部分组成的五元组$(Q, \Sigma, \delta, s, F)$:
- 状态集合Q; AC自动机上的每个点
- 字符集Σ; 字符集
- 状态转移函数$\delta: Q \times \Sigma \to Q$; AC自动机上的每条边
- 开始状态$s \in Q$; Trie树的根节点
- 接受状态集$F \subseteq Q$; 标记每个T结尾的点
- 可以借助AC自动机感性理解一下
- 通常DFA是DAG，但是AC自动机因为引入了$fail$指针所以有环

后缀自动机

- suffix automaton, SAM
- 字符串S的SAM是一个接受S所有后缀的最小DFA，因此：
- SAM是一个DAG，结点表示状态，边表示状态的转移
- 存在起始状态t_0，各个结点均可从t_0出发到达
- 边上标有字符，从一个点引出的边，字符均不同（函数的性质）
- 存在若干终止状态，如果从t_0出发到达了某个终止状态，路径上的字符连接起来一定是S的后缀；每个后缀同样可以用上面的方式得到
- 在所有满足性质的DFA中，SAM的状态数最少
- 存在性可以由后续的构造算法给出，我们分析过程中直接用...

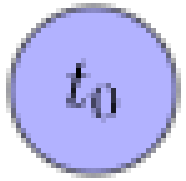
后缀自动机

- 因为从t_0出发可以接受所有后缀，因此可以用一个从t_0出发路径表示S的一个子串
- 为什么？
- 在接受后缀的中途停下，即可得到一个后缀的前缀...
- 不同的路径唯一对应一个本质不同的子串
- 为什么？
- 到达某个状态的路径不止一条，因此一个状态对应一些子串组成的集合

后缀自动机

- 以下是一些SAM的例子，蓝色表示起始状态，绿色表示终止状态
- 请大家注意观察是否接受S的每一个后缀，并表示出所有子串

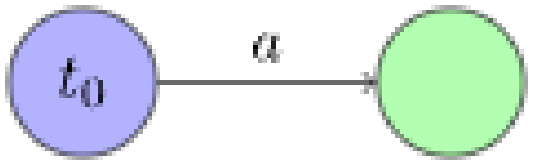
对于字符串 $s = \varnothing$：



后缀自动机

- 以下是一些SAM的例子，蓝色表示起始状态，绿色表示终止状态
- 请大家注意观察是否接受S的每一个后缀，并表示出所有子串

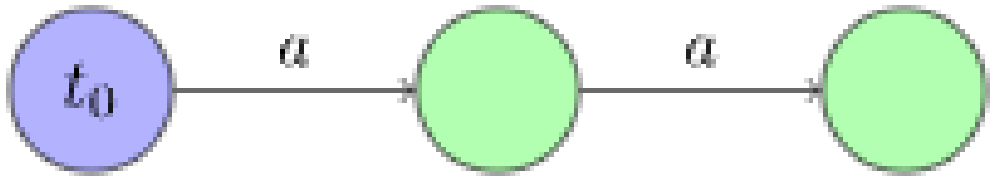
对于字符串 $s = \mathtt{a}$:



后缀自动机

- 以下是一些SAM的例子，蓝色表示起始状态，绿色表示终止状态
- 请大家注意观察是否接受S的每一个后缀，并表示出所有子串

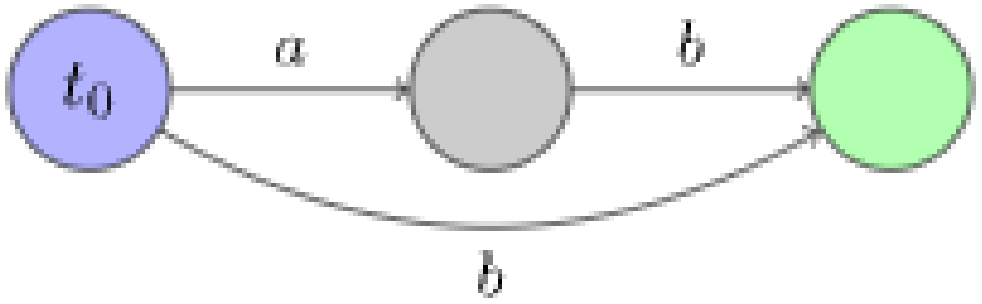
对于字符串 $s = \texttt{aa}$:



后缀自动机

- 以下是一些SAM的例子，蓝色表示起始状态，绿色表示终止状态
- 请大家注意观察是否接受S的每一个后缀，并表示出所有子串

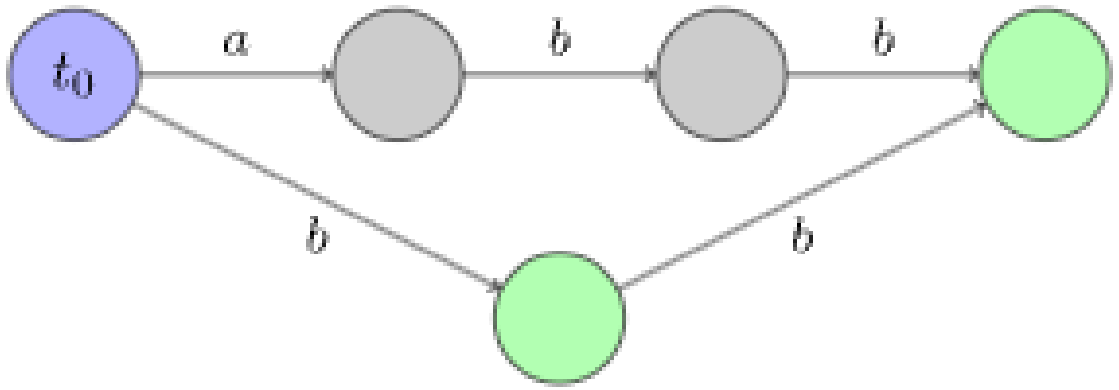
对于字符串 $s = \mathbf{ab}$:



后缀自动机

- 以下是一些SAM的例子，蓝色表示起始状态，绿色表示终止状态
- 请大家注意观察是否接受S的每一个后缀，并表示出所有子串

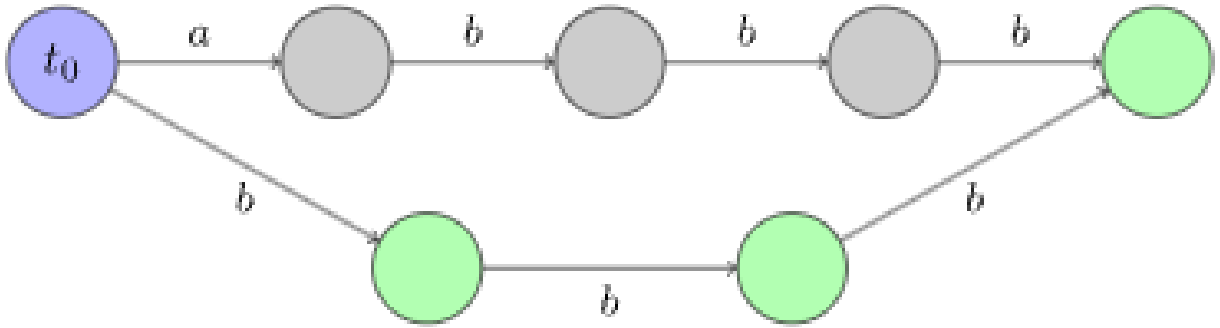
对于字符串 $s = \mathbf{abb}$:



后缀自动机

- 以下是一些SAM的例子，蓝色表示起始状态，绿色表示终止状态
- 请大家注意观察是否接受S的每一个后缀，并表示出所有子串

对于字符串 $s = \mathbf{abbb}$:



后缀自动机

- 在描述构造SAM的算法之前，需要理解几个重要概念

$endpos$

- 结束位置$endpos$集合，也有人称为$right$集合
- 考虑S的任意非空子串T，$endpos(T)$表示在S中T的所有结束位置
- 例如$S = abcbc,\ T = bc$，有$endpos(bc) = \{3,5\}$
- 请大家求出$S = aabab$所有子串T及对应的$endpos(T)$，并观察：
- 1.$endpos$集合相同的字符串之间的关系
- 2.不同$endpos$集合之间的关系

$endpos$

- **引理一**：对于S的两个非空子串$u, w(|u| < |w|)$，两者$endpos$集合相同，当且仅当u在S中的每次出现时，都是以w后缀的形式出现。
- 证明比较显然
- **引理二**：对于S的两个非空$u, w(|u| < |w|)$，要么$endpos$不交；要么$endpos(w) \subseteq endpos(u)$，此时$u$是$w$的后缀
- 当u是w的后缀时，w出现的位置u肯定也能出现
- **引理三**：将$endpos$相同的子串分入同一个等价类，每一个等价类内部的串长度是一个连续区间，并且短的串一定是长的串的后缀
- 考虑$minL$和$maxL$之间的长度，不可能$endpos$不等于它俩的

$endpos$

- 在SAM中，每个点对应一个$endpos$等价类
- 为什么？
- 因为可以走相同长度的步数变成后缀
- 因此走到同一个状态的所有路径，长度处于一个区间，且短的串为长的串的后缀

$link$

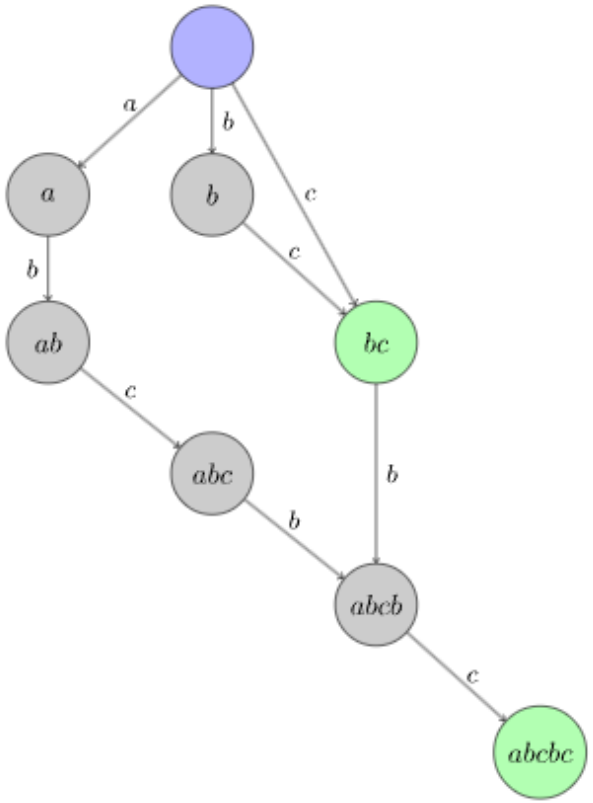
- 后缀链接$link$
- 考虑SAM中的状态v对应的$endpos$等价类，取其中最长的一个，设为w，等价类中其它串均为w的后缀
- 因为等价类中串的长度为一个区间，因此存在w的后缀（至少是一个空串）不在当前等价类里，记最长的那个后缀所在的等价类为t，$link(v) = t$
- 方便起见规定$endpos(t_0) = \{0,1,2,\dots,n\}$

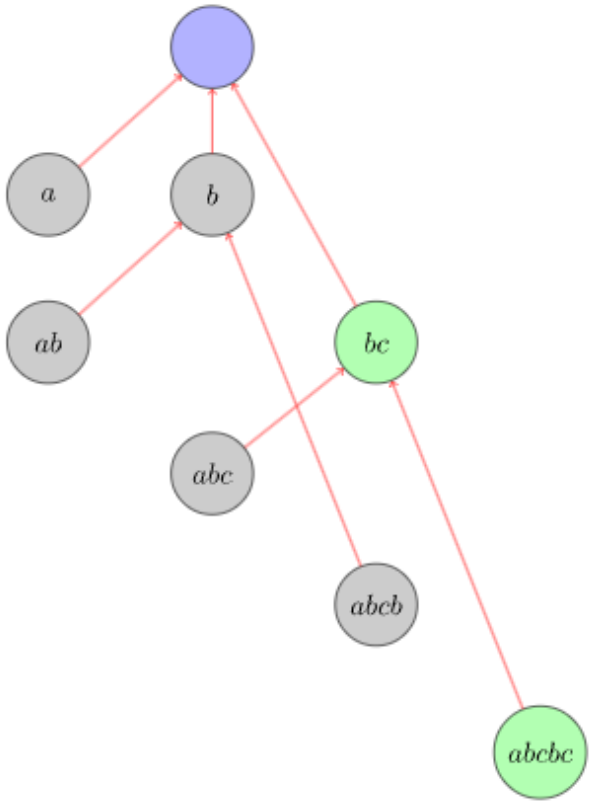
$link$

- **引理四**：所有后缀链接构成一棵根节点为t_0的树
- 每次跳$link$，等价类的长度区间在左移，最终一定移到空串上
- **引理五**：通过$endpos$集合构造的树（规则：祖先节点的集合一定包含后代的集合），与$link$构造的树相同
- 比较显然$link$指向的集合是包含当前集合当中最小的一个

小结

以下是对字符串 abcbc 构造 SAM 时产生的后缀链接树的一个 **例子**，节点被标记为对应等价类中最长的子串。





写一下$link$树上每个点的$endpos$以及所有子串

小结

- 小结一下并引入一些记号:
- S的子串可以根据$endpos$划分为若干等价类
- SAM由起始状态t_0和每一个$endpos$对应的状态组成
- 对于每个状态v，其包含若干个子串，记最长的长度为$len(v)$，最短的长度为$minlen(v)$，子串长度均在区间$[minlen(v), len(v)]$内，且均为最长子串的后缀
- 对应每个状态v，$link(v)$满足$minlen(v) = len(link(v)) + 1$，且$link(v)$包含的所有子串均是$v$中子串的后缀
- 从任意状态v沿着$link$往上跳，总会到达t_0，并且能得到若干不相交，且并起来为$[0, len(v)]$的区间

构造算法

- 构造算法是在线的，我们依次加入S的每个字符，并在每一步中对应地维护SAM
- 构造过程中只保存每个状态的边、len、$link$
- 初始SAM只有起始状态t_0，编号为1，$len[1] = link[1] = 0$，其中0是虚拟状态

构造算法

- 假设我们已经构造好了S的SAM，这时构造$S+c$的SAM
- 思考加入末尾的字符后，我们需要多出哪些东西
- 最直观的就是多出了若干没有出现过的后缀（至少$S+c$没出现过）
- 维护$last$表示添加字符c之前，S所处的状态
- 首先创建新的状态cur，$len(cur)=len(last)+1$
- 为什么？
- cur需要包含$S+c$

构造算法

- 从$last$开始一直沿着$link$跳，将所有没有c转移的状态，都连向cur
- 为什么？
- 因为$link$一直跳，$endpos$里面总会有n，原先状态没有c转移，表示对应的新后缀原先没有出现过
- 当跳到某个状态存在c转移，停下来，记该点为p
- 如果不存在这样的p，$link(cur) = t_0$，然后退出
- 为什么？
- 说明S中不包含任何c，否则至少t_0存在c的转移

构造算法

- 现在找到了一个状态p，通过c转移到q
- 说明正在尝试向SAM添加已经存在的字符串，设最长那个为$x+c$
- 需要想办法把$link(cur)$连到$x+c$对应的状态，并且$x+c$为该状态最长的串
- 显然q包含$x+c$，但什么情况下$x+c$是q中最长的串？
- $len(p)+1=len(q)$
- 如果$len(p)+1=len(q)$，则$link(cur)=q$，然后退出

构造算法

- 如果$len(p)+1<len(q)$，说明q中最长的串不是$x+c$，并且SAM中不存在以$x+c$为最长串的状态
- 但是$x+c$不能再属于q了，因为$x+c$的$endpos$里面有$n+1$
- 只能将q拆开，且第一个子状态的长度为$len(p)+1$
- 拆的时候对q进行复制，产生新状态$clone$
  $clone$需要复制q的所有转移和$link$，$len(clone)=len(p)+1$
- 然后设置$link(cur)=link(q)=clone$
- 最后还要将部分原先c转移到q的边，重定向到$clone$，从p开始沿着$link$跳即可，直到跳到虚拟节点或者c转移不是q

时空复杂度

- 时空复杂度?
- 如果把字符集大小$|\Sigma|$看作常数，时空复杂度均为$O(n)$
- 否则根据存边的方式不同，复杂度不同
- 如果用$|\Sigma|$的数组存边，时间复杂度$O(n)$，空间复杂度$O(|\Sigma|n)$
- 如果用map存边，时间复杂度$O(nlog|\Sigma|)$，空间复杂度$O(n)$
- 具体证明及实现参照论文