Summary of Suffix Automaton

2017-04-11 00:00:00

Definition

后缀自动机（以下简称 SAM）是一个典型的有限状态自动机，定义在给定的字符集上，可识别状态为一个字符串的所有子串，可以证明这个自动机的状态数和转移数都是线性的，常用于处理子串出现次数、字典序相关的字符串问题、以及要求强制在线的字符串匹配问题。

一些定义：

- $link(s)$ 状态 s 的后缀链接状态
- $len(s)$ 状态 s 的最长表示长度
- $trans(s, c)$ 在状态 s 下加入字符 c 能到达的转移状态
- $endpos(s)$ 状态 s 所对应的字符串出现在原串中的位置集合

Status in SAM

与一般自动机不同，在 SAM 中，一个状态对应的是一个字符串的集合，而且这个集合是原串的一段长度连续的子串。同时后缀自动机的状态有许多性质，考虑 trans 边的时候它是一个 DAG，而考虑 link 边的时候它又成了一棵树（事实上，这就是俗称的后缀树），本文接下来的部分将从这两个方面来讨论 SAM 的一些性质。

Trans -> DAG

$trans$ 边的定义与一般的有限状态自动机类似，表示一种合法的状态转移关系，简而言之就是在一个字符串的末尾添加一个字符，同时使得新的字符串是一个合法的串。

在 SAM 中，这个性质仍然成立但是要注意的是因为一个状态对应的不只有一个字符串，所以一个状态的 trans 状态**不一定恰好是这个状态的转移状态，还包括其他的状态转移到的状态**。

性质：不同状态的 endpos 集合必定不同。理由：如果两个状态的 endpos 集合是相同的，那么它们的后继状态就是相同的，那么在 DAG 上，这两个点就可以合并从而减少状态数。

Link -> Tree

前面已经提到，SAM 中的一个状态的 endpos 集合是固定的，在这种情况下满足条件的子串就是一段长度连续的子串，$link$ 边的定义就是**满足表示的子串集合恰好是当前状态所有子串的后缀**的状态中，len **值最大**的那个。

那么我们就不难发现，一个状态和它的 $link$ 之间的 endpos 集合恰好是包含关系。事实上，因为后缀自动机的一个状态表示的所有子串 endpos 集合相同，所以**任意两个状态的 endpos 集合之间仅存在不相交关系和包含关系**。

Construction of SAM

后缀自动机的构建采用增量法，假设当前已经处理好了原串的一个前缀的 SAM，那么新加入的状态就是前缀的任意一个后缀加入当前字符所产生的新的子串。

那么我们可以沿着 $link$ 边不断地往回跳，顺次遍历前缀的每一个后缀，并依次考虑加入新字符后的转移，如果不存在这种转移，则说明这个子串是第一次出现，添加这个转移指向新建节点即可。如果存在这个转移而且转移得到的状态的 len 恰好在原状态的基础上加了 1，那么说明这个串独立地作为一个状态**在原前缀串中出现过并且它的所有后缀都曾出现过**，直接将新

点的 *link* 指向这个点即可。否则说明这个串**出现过但是和别的一些串合并在一起**，现在这个串的 endpos 集合发生了改变（加上了最后一次出现的位置），所以需要一个新的点来表示这个串，然后把原来的和这个串之前所在状态相连的转移接到这个点上，剩下的点不用管（想一想，为什么）。

讲了这么多其实代码很好写：

```
void extend(int c) {
    int p = lst, cur = newnode(len[p]+1);

    while(p && !trans[p][c]) trans[p][c] = cur, p = link[p];

    if(!p) link[cur] = 1;
    else {
        int q = trans[p][c];
        if(len[q] == len[p] + 1) link[cur] = q;
        else {
            int nq = newnode(len[p]+1);$
            memcpy(trans[nq], trans[q], sizeof trans[q]);
            link[nq] = link[q];
            link[cur] = link[q] = nq;
            while(p && trans[p][c] == q) trans[p][c] = nq, p = link[p];
        }
    }lst = cur;
}
```