

后缀自动机

后缀自动机[suffix-automaton, SAM]用一种高压缩的方式存储了文本串的所有子串信息，其中最重要的信息是状态结点转移和后缀连接。下图表示字符串banana的后缀自动机，其中红色虚线代表后缀连接。



关于后缀自动机的一些性质可以从上面的实例中得以体现：

- 每条从 S 开始到任意结点结束的路径都构成了文本串的一个子串，所有从 S 开始的路径构成了文本串的所有子串集。
- 每条从 S 开始到终点结点（图中蓝色结点）的路径构成了文本串的一个后缀，同样的，所有这种路径构成了文本串的所有后缀集。
- 每个结点代表了唯一的一种 **endpos** 等价类，其意义是：该结点包含的所有子串（也即从 S 开始到该结点的路径组成的所有子串）在原文本串中有相同的结束位置集。比如图中对于 8 号结点 $\text{endpos}(\text{"an"}) = \text{endpos}(\text{"n"}) = \{3, 5\}$
- 每个结点包含的子串构成了一个连续的长度区间。我们令 $\text{maxlen}(u)$ 表示结点 u 的子串中最长的长度， $\text{longest}(u)$ 表示最长的子串，则 u 中所有的子串都是 $\text{longest}(u)$ 的后缀。
- 后缀连接的意义如下：结点 u 的后缀连接为不在 u 中的 $\text{longest}(u)$ 的最长的后缀所在的结点。也就是说设 $\text{minlen}(u)$ 为结点 u 的子串中最短的长度，那么 $\text{longest}(u)$ 的长度为 $\text{minlen}(u)-1$ 的后缀就不在 u 中了，假设在 v 中，那么 u 的后缀连接 $\text{link}[u]=v$ 显然有 **$\text{minlen}(u) = \text{maxlen}(\text{link}[u]) + 1$** （这个非常重要）。
- 沿着一个结点 u 的后缀路径遍历所有该路径上的结点，可以遍历到 $\text{longest}(u)$ 的所有后缀。
- 沿着最后一个建立的结点的后缀路径遍历所有该路径上的结点，可以得到所有的终点结点。

接下来说一下后缀自动机的构造方法。

假设当前已经构造好了 $S[1\dots n]$ 的后缀自动机，当我们新增添 $c=S[n+1]$ 的时候，需要考虑 $n+1$ 个新的后缀。设上一次新增字符增添的结点为 last ，这次肯定要新增加一个新的结点 cur ，否则字符串 $S[1\dots n+1]$ 就无法表示。令 p 沿着 last 的后缀路径遍历，如果 $\text{trans}[p][c]$ （转移）不存在，那么令 $\text{trans}[p][c]=\text{cur}$ ，否则跳出。如果一直到 S 仍然转移不存在，最后令 $\text{link}[\text{cur}]=S$ ，否则假设第一个存在 $\text{trans}[p][c]$ 的结点是 u ， $\text{trans}[u][c]=v$ ，就要分两种情况讨论了：

1. 如果 $\text{maxlen}(u) + 1 = \text{maxlen}(v)$ ，说明 v 中所有子串都是 $S[1\dots n+1]$ 的后缀，故直接令 $\text{link}[\text{cur}] = v$
2. 否则，新增添一个结点 clone ，令 $\text{trans}[\text{clone}] = \text{trans}[v]$ ，然后把从 u 开始的后缀路径上所有 $\text{trans}[p][c] == v$ 的转移都改成 $\text{trans}[p][c] = \text{clone}$ ，保证 clone 的所有子串都是 $S[1\dots n+1]$ 的后缀， clone 继承 v 的后缀连接， $\text{link}[v] = \text{link}[\text{cur}] = \text{clone}$

构造代码如下（构造仅仅是 extend 那一部分，所有数组从 1 开始）

```
// 1号是S结点
const int maxn=1e6+5;
struct suffix_automaton
{
    int maxlen[maxn<<1],trans[maxn<<1][26],link[maxn<<1],tot=1,last=1;
    int endnum[maxn<<1],c[maxn<<1],a[maxn<<1];
    int e[maxn<<1];

    void extend(int c)
    {
```

```
int cur=++tot,p; endnum[cur]=1;
maxlen[cur]=maxlen[last]+1;
for(p=last;p && !trans[p][c];p=link[p]) trans[p][c]=cur;
if(!p) link[cur]=1;
else
{
    int q=trans[p][c];
    if(maxlen[q]==maxlen[p]+1) link[cur]=q;
    else
    {
        int clone=++tot;
        maxlen[clone]=maxlen[p]+1;
        memcpy(trans[clone],trans[q],sizeof(trans[q]));
        for(;p && trans[p][c]==q;p=link[p]) trans[p][c]=clone;
        link[clone]=link[q];
        link[q]=link[cur]=clone;
    }
}
last=cur;
}

void get_endpos_num() // count the end-position number of every
node(state)
{
    REP(i,1,tot) c[maxlen[i]]++;
    REP(i,1,tot) c[i]+=c[i-1];
    REP(i,1,tot) a[c[maxlen[i]]--]=i;
    REP_(i,tot,1) endnum[link[a[i]]]+=endnum[a[i]];
}

void get_end() // get the end node(state)
{
    for(int i=last;i!=1;i=link[i]) e[i]=1;
}

void build(char *s,char minc)
{
    for(int i=0;s[i];i++) extend(s[i]-minc);
    get_endpos_num();
    //get_end();
}
};
```

后缀自动机的时间复杂度和空间复杂度都是 $O(n)$ □

关于后缀自动机的一些常见应用□

- 完美取代AC自动机，多模式匹配问题直接对文本串建后缀自动机，然后每个模式串从S开始跑一遍

就知道是不是子串以及出现次数了。

- 子串出现次数。我们需要计算每个结点对应结束位置的个数 $endnum$ 每一个长的子串出现时，都对其更短的后缀有贡献，故（一定要忽略clone的结点）对每次新建结点出现次数赋值为1，然后在后缀连接图上拓扑序累加即可，或者可以按照每个结点的 $maxlen$ 排个序，然后从大到小累加也可（可以说明这样子满足拓扑序）。
- 本质不同子串个数 = $\sum_{i=2}^{\text{tot}}(maxlen[i]-maxlen[link[i]])$

一些题目

- **最长公共子串**：（先给其中一个子串建sam然后另一个在上面跑，不匹配的时候跳link我的代码
- **hdu6583 Typewriter** dp+sam 设 $f[i]$ 为打印到第 i 个字符的最少花费，则 $f[i]=\min\{f[i-1]+p, f[j]+q\}$ 其中 j 是最小的下标，使得子串 $s[j+1, \dots, i]$ 是 $s[1, \dots, j]$ 的子串，可以发现 j 随着 i 的增加而增加，所以用sam维护 $s[1, \dots, j]$ 的信息，每次增加 i 的时候保证 $s[j+1, \dots, i]$ 是 $s[1, \dots, j]$ 的子串）（我的代码

广义后缀自动机

就是可以处理多个字符串后缀信息的后缀自动机，据说是一种Trie+SAM的结构，但是以下的做法一般来说也没什么问题：

- 建立一个SAM每次插入一个字符串之后把last置为1

一些例题

- **洛谷 广义后缀自动机**：（这道题以上做法确实没问题）（我的代码

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

Permanent link: https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:too_low:%E5%90%8E%E7%BC%80%E8%87%AA%E5%8A%A8%E6%9C%BA&rev=1592533145

Last update: 2020/06/19 10:19