后缀自动机

2021-01-03 | 2022-08-20 | OI & ACM | 9
8.6k | 8 分钟

简介

字符串补完计划

后缀自动机

一些约定和定义

DFA 即确定性有限状态自动机，由五部分组成 $M=(\sum,Q,q_s,q_t,tr)$

$\sum$ 表示有限字符集，其中每个字符 c 表示一个输入符号

Q 表示一个有限状态集合

$q_s\in Q$ 称为初始集合

$q_t\in Q$ 称为终结状态集合

$tr(q,c)$ 称为状态转移函数

$L(q)$ 表示状态 q 能识别的串的集合

$L(M)$ 表示自动机 M 能识别的串的集合

称自动机 M 识别一个串当且仅当 $tr(q_s,S)\in q_t$

对于任意 $p,q\in Q$，称 p 和 q 为等价状态当且仅当 $tr(p,S)\in q_t\iff tr(p,S)\in q_t$

即 $L(p)=L(q)$ 成立，此时记作 $p\sim q$

SAM 的定义

字符串 S 的 SAM 是一个接受 S 的所有后缀的最小 DFA

换句话说

- SAM 是一张有向无环图，结点被称作 **状态**，边被称作状态间的 **转移**
- 图存在一个源点 q_s，称作 **初始状态**，其它各结点均可从 q_s 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均 **不同**
- 存在一个或多个 **终止状态**，如果我们从初始状态 q_s 出发，最终转移到了一个终止状态，则路径上的所有转移连接起来一定是字符串 S 的一个后缀，S 的每个后缀均可用一条从 q_s 到某个终止状态的路径构成
- 在所有满足上述条件的自动机中，SAM 的结点数是最少的

right 集合

定义：设 $T \in \sum*$，定义 T 在母串 S 的所有结束位置的右端点的集合如下：$right(T) = \{r | \exist 1 \le l \le r, T = S[l \cdots r]\}$

引理 1：对于任意串 $T \in \sum*$，有 $L(tr(q_s, T)) = \{suf_{r+1} | r \in right(T)\}$

引理 2：设 $T_1, T_2 \in \sum*$ 是任意两个串，则 $tr(q_s, T_1) \sim tr(q_s, T_2)$ 充分必要条件是 $right(T_1) = right(T_2)$，此时称这两个串等价，记作 $T_1 \sim^S T_2$

由于 SAM 的最小性，我们能够知道一个状态 q 对应一个 $right$ 集合，我们不妨令其为 R_q

同时我们知道一个状态 q 也对应多个串 T，且 $tr(q_s, T) = q$，$right(T) = R_q$，我们不妨称这样的 T 构成的集合为 S_q

注意到如果两个串等价，不妨设 $len(T_1) < len(T_2)$，那么我们知道 T_1 必然是 T_2 的后缀

那么我们注意到对于一个 $right$ 集合，不妨设其为 R，它所对应的串有什么性质呢

不难发现，R 所对应的串的长度应该是一个区间，设其为 $[minl_R, maxl_R]$

这里我们还能够想到一个问题，一个串的子串的个数是 $O(n^2)$ 的，但是其中有很多子串是等价的

或者换句话说，这 $O(n^2)$ 个子串按照 $right$ 集合分成若干个等价类

引理3：任取两个不同的状态 $p, q \in Q$，我们一定有下面三式之一成立

1. $R_p \cap R_q = \empty$
2. $R_p \subset R_q$
3. $R_q \subset R_p$

parent 树

定义：对于任意 $p, q \in Q$，我们称 p 是 q 的 $parent$ 状态当且仅当 R_p 是最小的真包含 R_q 的集合

另外定义 $R_{q_s} = \{0, 1, 2, \cdots, n\}$

引理 3：任何状态的 $parent$ 状态存在且唯一

也就是说，我们可以用根据 $parent$ 状态连出来一棵树

引理 4：若 p 是 q 的 $parent$ 状态，则 $maxl_p = minl_q - 1$

构造算法

```
struct SAM {
    int sz, L, l, nxt[26];
} T[Maxn]; int f[Maxn], top, last, rt;
void init_SAM() {
    for (int i = 1; i <= top; ++i) {
        T[i].sz = T[i].L = T[i].l = f[i] = 0;
        fill(T[i].nxt, T[i].nxt + 26, 0);
    }
    rt = last = top = 1;
    T[rt].L = T[rt].l = f[rt] = 0;
}

void extend(int ch) {
    int np = ++top, p = last; last = np;
    T[np].L = T[p].L + 1; T[np].sz = 1;
    while (p && !T[p].nxt[ch]) T[p].nxt[ch] = np, p = f[p];
    if (!p) return f[np] = rt, void();
    int q = T[p].nxt[ch];
```

```
        if (T[q].L - 1 == T[p].L) f[np] = q;
        else {
            int nq = ++top; T[nq].L = T[p].L + 1; f[nq] = f[q];
            for (int i = 0; i < 26; ++i) T[nq].nxt[i] = T[q].nxt[i];
            while (p && T[p].nxt[ch] == q) T[p].nxt[ch] = nq, p = f[p];
            f[np] = f[q] = nq;
        }
    }
```

性质

将 $extend$ 过程中新建的 np 节点所构成的链叫做主链

1. $right$ 集合的大小等于其在 $parent$ 树上的子树中在主链上的点的状态数

   证明：

   换一句话说，就是只有 np 节点会有贡献，那么其实我们只需要证明 nq 节点没有贡献就行了

   nq 节点相对于 q，$right$ 集合只是多了 $\{L+1\}$，而 $R_{np}=\{L+1\}$，并且 np 的 $parent$ 的状态是 nq

   所以 nq 的 $right$ 集合多出来的 $\{L+1\}$ 实际上是 np 的

2. pre_i 和 pre_j 的最长公共后缀的长度为 pre_i 和 pre_j 所对应的状态在 $parent$ 树上的 lca 的长度，如果将串反过来建 SAM，就能求得两个后缀的最长公共前缀

模板

基数排序

```
int tax[maxn], tp[Maxn];
void rsort(int n) {
    for (int i = 1; i <= n; ++i) tax[i] = 0;
    for (int i = 1; i <= top; ++i) ++tax[T[i].L];
    for (int i = 1; i <= n; ++i) tax[i] += tax[i - 1];
    for (int i = 1; i <= top; ++i) tp[tax[T[i].L]--] = i;
    for (int i = top, u = tp[i]; i > 1; u = tp[--i]) T[u].l = T[f[u]].L + 1;
}
```

字符串在 SAM 上匹配

```
int p = rt, len = 0, ans = 0;
for (int i = 1; i <= m; ++i) {
    int ch = t[i] - 'a';
    if (T[p].nxt[ch]) p = T[p].nxt[ch], ++len;
    else {
        while (p && !T[p].nxt[ch]) p = f[p];
        if (!p) p = rt, len = 0;
        else len = T[p].L + 1; p = T[p].nxt[ch];
    }
    ans = max(ans, len);
 } cout << ans << "\n";
```

应用

1. 本质不同的子串个数

   这个显然就是所有状态所对应的 $right$ 集合的大小的和

2. 求两个串的最长公共子串

   不妨令着两个串为 S 和 T

   我们用 S 建 SAM

   首先我们知道子串一定是某个前缀的后缀

   所以我们用 p 来表示对于 T 的 pre_i 的匹配 S 最长后缀所对应 S 的 SAM 的状态，len 表示匹配上的长度

   那么从 pre_i 转移到 pre_{i+1}，如果存在 $tr(p,T[i+1])$，则 $p=tr(p,T[i+1])$，且 len 加 1

   否则我们只需要在 SAM 上对于 p 不断跳 $parent$ 直到 $tr(p,T[i+1])$ 存在

   因为每次 len 最多加 1，且跳 $parent$ 会使 len 至少减 1，所以总的时间复杂度还是 $O(n)$

```
int p = rt, len = 0, ans = 0;
for (int i = 1; i <= m; ++i) {
    int ch = t[i] - 'a';
    if (T[p].nxt[ch]) p = T[p].nxt[ch], ++len;
    else {
        while (p && !T[p].nxt[ch]) p = f[p];
        if (!p) p = rt, len = 0;
        else len = T[p].L + 1, p = T[p].nxt[ch];
    }
    ans = max(ans, len);
 } cout << ans << "\n";
```

3. 第 k 小子串

   注意到子串为 SAM 上的一条路径，而 SAM 可以看成有向无环图

   所以第 k 小子串实际上就是第 k 小路径，预处理一下直接做即可

4.

广义后缀自动机

实际上我对广义后缀自动机的理解是建在 $Trie$ 上的后缀自动机

也就是说，广义后缀自动机能识别的后缀是 $Trie$ 上的后缀

构建广义后缀自动机有两种方法

1. 离线做法

   从 $Trie$ 的根开始 bfs，依次将遍历的节点插入 SAM，插入时的 $last$ 就等于父节点的 $last$

   这样可以保证插入的前缀的长度非降，所以可以直接套用一般构建 SAM 的方法即可

   可以证明，这样的复杂度为 $O(L \times \sum)$

2. 在线做法

或者说是 dfs，这样的复杂度为 $O(L \times \sum + g(T))$，其中 $g(T)$ 为 T 的所有叶子节点的深度和

这样无法保证插入的前缀的长度非降，所以可能会导致要插入的新节点要成为之前的某个节点的父亲

这其实和构建 SAM 分叉的情况一样，特判一下即可

但实际上不需要真正的建出 $Trie$ 每次插入一个串从根走即可

这个是不建出 $Trie$ 的在线做法

其实建不建 $Trie$ 的在线做法都是一样的

```
int extend(int p, int ch) {
    if (T[p].nxt[ch]) {
        int q = T[p].nxt[ch], nq;
        if (T[q].L - 1 == T[p].L) return q;
        nq = ++top; T[nq].L = T[p].L + 1; f[nq] = f[q];
        memcpy(T[nq].nxt, T[q].nxt, sizeof T[q].nxt);
        while (p && T[p].nxt[ch] == q) T[p].nxt[ch] = nq, p = f[p];
        f[q] = nq; return nq;
    }
    int np = ++top; T[np].L = T[p].L + 1;
    while (p && !T[p].nxt[ch]) T[p].nxt[ch] = np, p = f[p];
    if (!p) return (f[np] = rt, np);
    int q = T[p].nxt[ch];
    if (T[q].L - 1 == T[p].L) f[np] = q;
    else {
        int nq = ++top; T[nq].L = T[p].L + 1;
        f[nq] = f[q]; memcpy(T[nq].nxt, T[q].nxt, sizeof T[q].nxt);
        while (p && T[p].nxt[ch] == q) T[p].nxt[ch] = nq, p = f[p];
        f[np] = f[q] = nq;
    }
    return np;
}

void insert(char *s) {
    int l = strlen(s), p = rt;
    for (int i = 0; i < l; ++i) p = extend(p, s[i] - 'a');
}

```

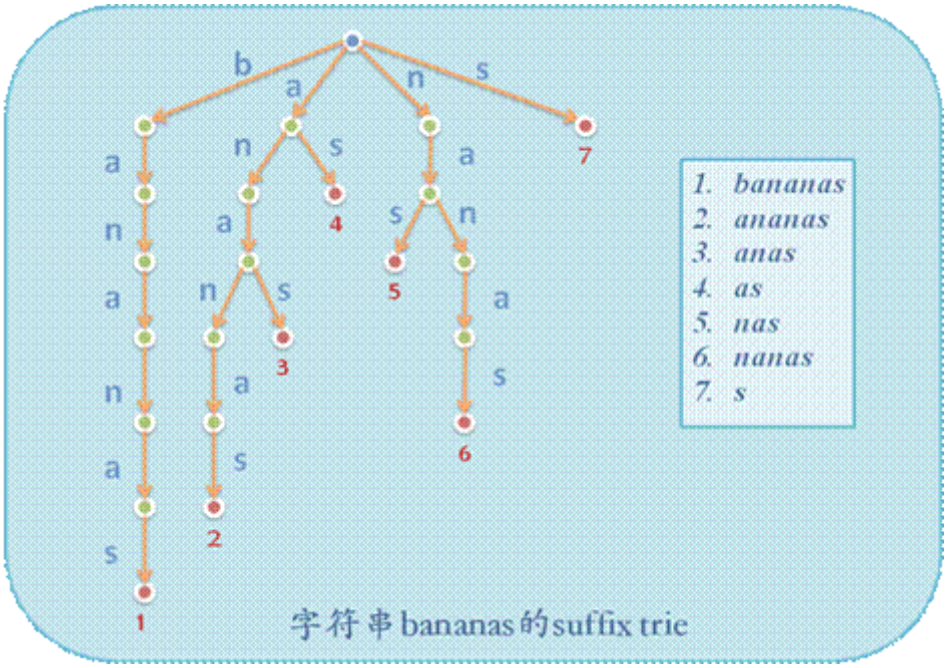
这个是不建 $Trie$ 的离线做法

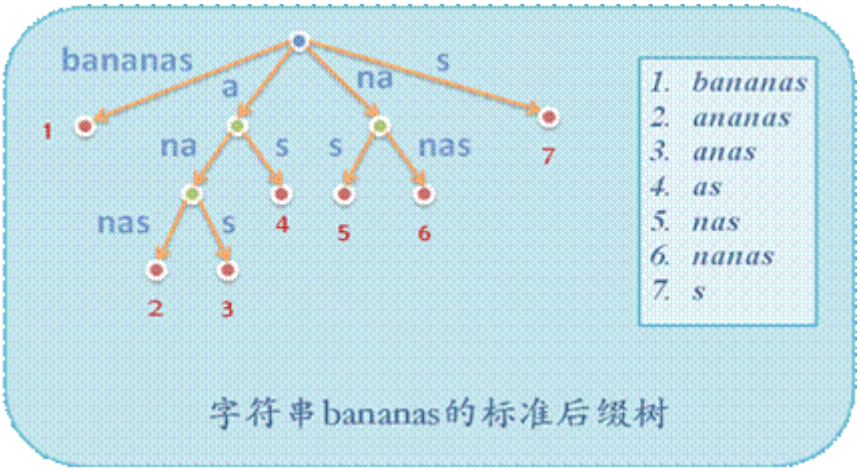
```
1
```

建 $Trie$ 的离线做法的 $extend$ 和普通的是一样的

后缀树

实际上后缀树就是将所有后缀依次插入 $Trie$ 里，然后把没有分叉的边都压缩一下，实际上就是一个压缩 $Trie$





(图片来自网络)

那么我们考虑一下后缀自动机和后缀树有什么关系

这里有一个结论，反串的 SAM 的 $parent$ 树是原串的后缀树

那么现在的问题是 SAM 建出 $parent$ 树的转移边上是没有字母的

通过观察容易得到这一串字母就是 $S_{rev}[T[u].pos-T[u].l+1]$ 到 $S_{rev}[T[u].pos-T[u].L+1]$，假设是 $f[u]$ 连到 u

```
int tax[maxn], tp[Maxn];
void rsort(int n) {
    for (int i = 1; i <= n; ++i) tax[i] = 0;
    for (int i = 1; i <= top; ++i) ++tax[T[i].L];
    for (int i = 1; i <= n; ++i) tax[i] += tax[i - 1];
    for (int i = 1; i <= top; ++i) tp[tax[T[i].L]--] = i;
    for (int i = top, u = tp[i]; i > 1; u = tp[--i]) {
        T[f[u]].pos = max(T[f[u]].pos, T[u].pos);
        T[u].l = T[f[u]].L + 1;
        add_edge(f[u], u, s[T[u].pos - T[u].l + 1] - 'a');
    }
}
```

例题

1. 简要题意：给定一个字符串 S 和 k，求 S 的本质不同/相同的第 k 小子串

   $|S| \leq 5 \times 10^5$

   简要题解：注意到任何一个子串都是 SAM 上的一条路径，而 SAM 可以看成 DAG

   所以第 k 小子串实际上就是第 k 小路径，预处理一下直接做即可

   Luogu P3975 [TJOI2015]弦论

2. 简要题意：每次添加一个字符，求添加一个字符后有多少新增加的本质不同的子串

   $n \leq 10^5$

   简要题解：每次新增的本质不同的子串的个数为 $T[np].L-T[np].l+1$

   Luogu P4070 [SDOI2016]生成魔咒

3. 简要题意：给定 n 个模板串和 m 个查询串，对于每个查询串查询它在几个模板串中作为子串出现

$len \leq 3.6 \times 10^5$

简要题解：

我们对于所有模板串建广义 SAM，然后用线段树合并求出 SAM 上每个节点是多少个模板串的子串

然后对于每个询问串，只需要将其放到 SAM 上跑，然后求出最后到达的节点是多少个模板串的子串即可

SPOJ 8093 JZPGYZ - Sevenk Love Oimaster

4. 简要题意：给定两个字符串 S, T 和 q 次询问，每次询问 $S[l \cdots r]$ 和 T 的最长公共子串长度

$|S|, |T| \leq 2 \times 10^5, q \leq 2 \times 10^5$

简要题解：我们考虑对于 T 建 SAM，然后对于所有 $pre(S, i)$ 我们求出于 T 的最长公共后缀的长度 len_i，这个过程就是拿 S 在 T 的 SAM 上做匹配

对于一次询问，根据 len_i，我们能够得到 $pre(S, i)$ 的与 T 匹配的最长公共后缀的左端点 $L_i = i - len_i + 1$，

我们的答案是 $max\{i - max\{L_i, l\} + 1\}, i \in [l, r]$，注意到 L_i 是单增的，所以可以二分出 L_i 比 l 大的位置

Luogu P6640 [BJOI2020] 封印

5. 简要题意：给定一个串 S，对于它的每个前缀求这个前缀的字典序最大的子串

$|S| \leq 10^6$

简要题解：我们首先考虑一个暴力，我们按字典序从大到小枚举所有本质不同的子串，然后找到这个子串的最早出现位置，那么从这个位置向后一直延伸直到出现一个在之前被标记过的位置为止，这些位置的答案都是这个子串

注意到 SAM 上的每一条路径都代表一个子串，那么我们直接在 SAM 上按照字典序来 dfs 即可，同时记录目前覆盖的位置，剪枝就是如果走到的位置已经大于等于已经覆盖的位置则可以直接退出，这样可以保证每个点只会经过最多一次，那么我们的复杂度就是构造 SAM 的复杂度 $O(\sum L)$

另外还有一个后缀树的做法，对于后缀树上一个节点，注意到这个节点对应的答案是一个区间，那么我们考虑按字典序进行 dfs 后，从字典序大的节点开始赋值，每次相当于合并一个区间，后面的合并不能影响前面的操作，那么可以使用并查集来操作，另外后缀树上每个节点的出现位置我们只需要记第一个即可，因为每个节点所代表的串只有可能在第一个位置成为答案，时间复杂度 $O(L(\sum +\alpha))$

The 2021 ICPC Asia Shenyang Regional Contest M String Problem

6. 简要题意：给定一个串 S，现在有 m 次询问，每次询问本质不同的第 k 小的子串，这里的字典序是长度不同，则长度小的字典序小，长度相同按照从前向后按位判断

   $|S|, m \leq 10^6$

   简要题解：首先我们将问题转换为求长度为定值的第 k 小子串

   我们考虑建出原串的后缀树，然后我们在 dfs 的时候按照边从小到大进行 dfs，这样每个点的 dfs 序就是字典序

   我们考虑将询问按长度离线，后缀树上每个点代表长度为一个区间的子串，那么我们可以做一个差分，在长度左端点加入，在长度右端点删除，这里我们用权值线段树维护第 k 小的字典序即可

   时间复杂度 $O(L \log L)$，我的代码 $vector$ 用的有点多，被卡空间了

   The 2021 CCPC Guilin Onsite (Grand Prix of EDG) J Suffix Automaton

7. 简要题意：给定一个长度为 n 的模板串 S 和 m 个查询串 T_i，现在对 S 做 m 次循环左移，令 Q_i 表示第 i 次循环左移的结果，$Q_i = S_iS_{i+1} \cdots S_nS_1S_2 \cdots S_{i-2}S_{i-1}$，同时令 Q_i 的贡献为 Q_i 的长度最长的一个子串的长度，满足这个子串也是某个 T_i 的子串，现在要求所有 Q_i 的贡献的最小值

$n, m, |S|, \sum |T_i| \leq 10^5$

简要题解：我们考虑首先将 S 复制一遍，这样以 S_i 开始的长度为 n 的子串就是 Q_i，然后我们对所有 T 建广义后缀自动机，这样我们可以在枚举 S 前缀的同时得到当前前缀的最长后缀，满足这个后缀是某个 T 的子串

我们考虑二分答案，那么对于每个枚举出的前缀的后缀，我们发现满足答案的 Q_i 的 i 是一个区间，那么我们直接做一个区间覆盖即可判断，这里的所有区间覆盖可以直接通过差分来做，时间复杂度 $O(n \log n)$

2021-2022 ACM-ICPC Brazil Subregional Programming Contest B Beautiful Words

8. 简要题意：给定一个长度为 n 的字符串 S，现在有 m 次询问，每次询问给定区间 $[l, r]$，求 $S[l..r]$ 有多少本质不同的子串

$n \leq 10^5, m \leq 2 \times 10^5$

简要题解：我们首先考虑一个弱化的问题，每次求结束位置在 $[l, r]$ 的本质不同的子串，我们对 S 建立后缀自动机，然后我们找到 1 到 n 所对应的 n 个 np 节点，那么区间 $[l, r]$ 的查询相当于是查询 $[l, r]$ 的 np 节点在 $parent$ 树上树链的并，对于这个问题我们还是考虑离线，我们考虑做扫描线，将树上每个节点的值挂在子树内最大 np 节点上，那么每次查询相当于是区间查询，每次修改相当于是将一个点到根的路径上的所有节点重新染色，注意到对于同一个颜色段的操作相同，而且这个操作即是 LCT 的 $access$ 操作，所以我们可以用 LCT 维护同色链，用树状数组维护权值，总时间复杂度 $O(n \log^2 n + m \log n)$

然后我们回到这个问题，我们现在的限制不只是结束位置在 $[l, r]$，还要保证起始位置也在 $[l, r]$，但其实操作类似，对于 $parent$ 树上的点，不妨设这个点所表示的串长在 $[a, b]$ 以及当前扫描线扫到 r，我们只需要将这个点的每个串的贡献挂在 $[r - a + 1, r - b + 1]$ 即可，同时在 $parent$ 树上，树链所表示的串长是一个连续区间，可以维护，那么我们现在每次修改相当于是区间加等差数列单点查询，我们可以将其转换成区间加区间查询，这样我们仍然可以使用树状数组来维护，时间复杂度 $O(n \log^2 n + m \log n)$

Luogu P6292 区间本质不同子串个数

Tech # 后缀自动机

CF 1469E A Bit Similar

CF 1400E Clear the Multiset