字符串进阶

张博凯

天津大学

2023 年 8 月 9 日

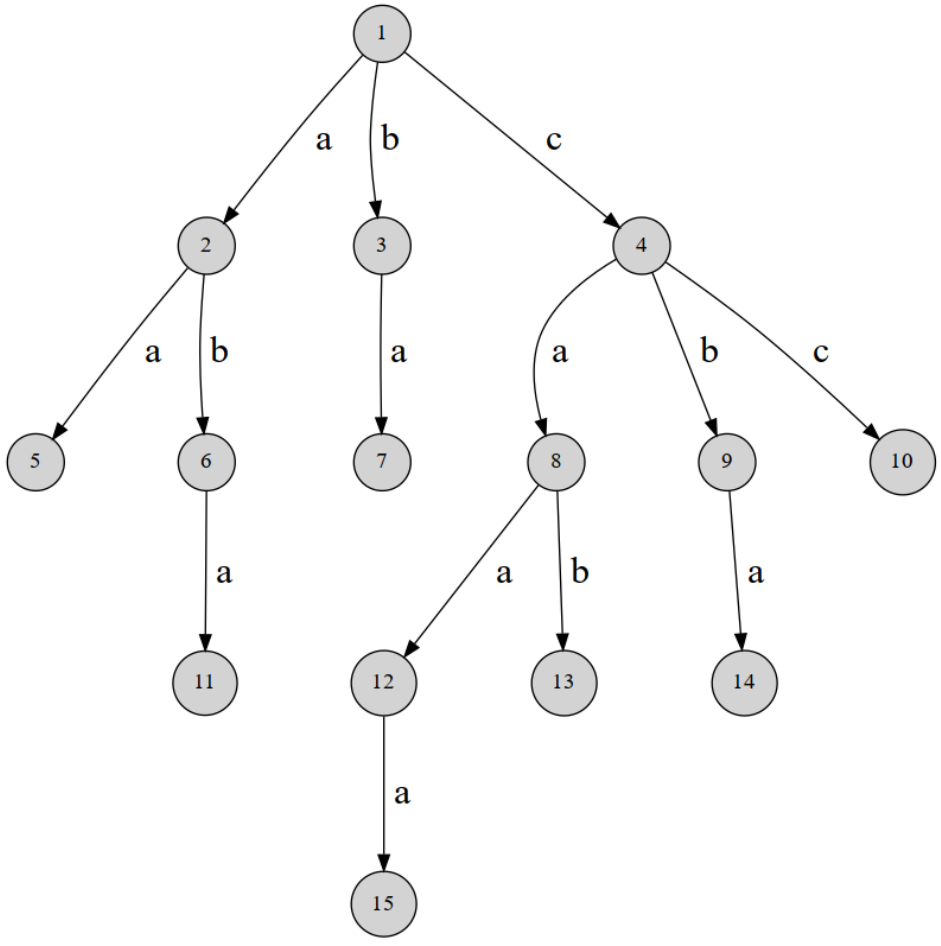
目录

目录

trie 树

Description

trie 树又叫字典树，它是一棵树，其中每条边都代表一个字符。每个节点表示从根到这个节点的路径上的字符连起来构成的字符串。



Trie

trie 树

建 trie

给定一些字符串，可以使用这些字符串建立一个 trie 树。我们一个个字符串地插入。每插入一个字符串，置当前节点为根，顺序枚举当前字符串中的字符，每次查看当前节点有没有当前字符对应的出边。若有则直接顺着已有的边走下去，否则新建一个节点，将当前节点置为新建节点。

简单应用

给定一个字符串集合 S。有 q 次询问，每次给定一个字符串 t，询问 t 是否为集合 S 中某个字符串的前缀。
直接建 trie，每次查询时在 trie 树上跑字符串 t，如果遇到了不存在的边走不通则输出 NO。否则输出 YES。

目录

AC 自动机

Description

AC 自动机是解决多串匹配的利器。也可以说它是 trie 树上的 KMP。一个经典的 AC 自动机问题是：给定一个字符串集合 S（字典）和一个字符串 t（文章），求字典中每个字符串在文章中出现次数之和。而 KMP 解决的经典模式匹配问题可以看作这个问题的简化版。

自动机

DFA，即确定有限自动机，是由五个要素组成的五元组。

1. 状态集合 Q
2. 字符集 Σ
3. 状态转移函数 δ
4. 一个开始状态 s
5. 一个接收状态集合 F

可以看作一张有向图。每个状态就是点，状态转移函数就是边。每个节点都有 $|\Sigma|$ 条出边，对应着当前状态接收到每个字符应该转移到的目标状态。

AC 自动机

AC 自动机

AC 自动机是这样的自动机：状态集合是字典中所有字符串的所有前缀的集合（也就是 trie 树的节点）。状态转移函数满足这样的性质：在自动机上运行一个字符串 t，会到达一个状态，这个状态代表的字符串是 t 最长的后缀，使得这个状态在状态集合中，即它既是 t 的后缀，又是字典中某字符串的前缀，同时尽可能长。

AC 自动机

AC 自动机的构建

由于 AC 自动机的状态集合就是 trie 树的节点，且原 trie 树上的边显然是 AC 自动机的转移边，那么首先建 trie，然后在 trie 树上新增转移边来构建 AC 自动机。设当前节点为 a，则我们主要关心 a 的原 trie 树上没有的出边。这就是失配的情况，即新添加的字符 c 不能在 trie 树上继续匹配，必须减少匹配的长度。这时，与 KMP 类似，我们求出一个 fail 数组。每个节点有一个 fail 指针，它指向另一个状态节点，它是当前节点的最长后缀，使得这个后缀在状态集合中。与 KMP 的分析类似，我们只需不断跳 fail，直到到达一个节点 b，它有对应字符 c 的出边指向 d，那么就把 a 向 d 连一条字符 c 的转移边。

AC 自动机

优化

每次都不断跳 fail 的复杂度无法接受。考虑如果节点 a 在 trie 树上没有字符 c 的出边，fail 指向 b。若 b 在 trie 树上仍然没有字符 c 的出边，但它之间已经跳过 fail 得到了字符 c 的转移边指向 d，那么实际上 a 的字符 c 转移边最终也会指向 d。这样就不需要跳 fail 了。于是，只需保证在处理一个节点的转移边时，其 fail 指针指向的节点已经处理过，就能保证 O(1) 得到转移边。由于 fail 边指向的节点深度一定比当前节点小，于是按 bfs 的顺序处理所有节点的转移边即可做到线性。fail 指针也可以类似地通过之前已经求结果的信息来 O(1) 得到。

```
void build(){
    l=1;r=0;
    for(int i=0;i<26;i++)if(trie[0][i])q[++r]=trie[0][i];
    for(;l<=r;l++){
        int x=q[l];
        for(int i=0;i<26;i++){
            int &y=trie[x][i];
            if(!y)y=trie[fail[x]][i];
            else fail[y]=trie[fail[x]][i],q[++r]=y;
        }
    }
}
```

AC 自动机

多串匹配

对于前面提到的多串匹配问题，我们把字典建成 AC 自动机，使用文章在上面运行。注意到运行到字典的一个前缀时，此时匹配到了字典中的字符串 s 当且仅当当前运行到的状态节点在代表 s 的状态节点的 fail 树的子树中。于是，我们预处理每个状态节点在 fail 树中到根的链中包含多少个代表字典中字符串的节点，即为运行到这个节点时，匹配到的单词的个数。然后在 AC 自动机上跑，不断累积贡献即可。

AC 自动机上 DP

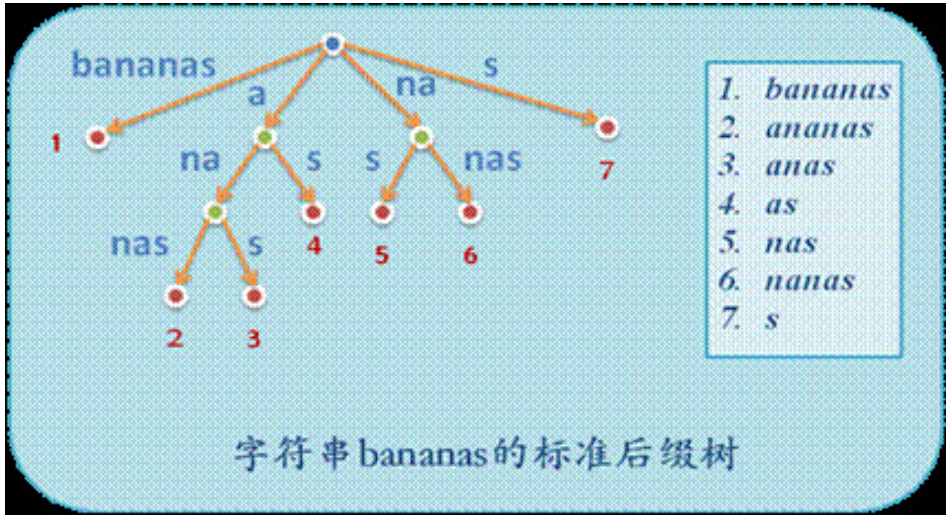
题目：给定一个字符串集合，求一个长度为 n 的字符串 s，使集合中每个字符串在 s 中出现次数之和最大。套路：定义 dp[i][j]，表示长度为 i 的字符串，当前运行到 AC 自动机的状态 j，的最大答案。转移时枚举状态，枚举下一个字符，AC 自动机上计算贡献转移来 DP 即可。

目录

后缀树

Description

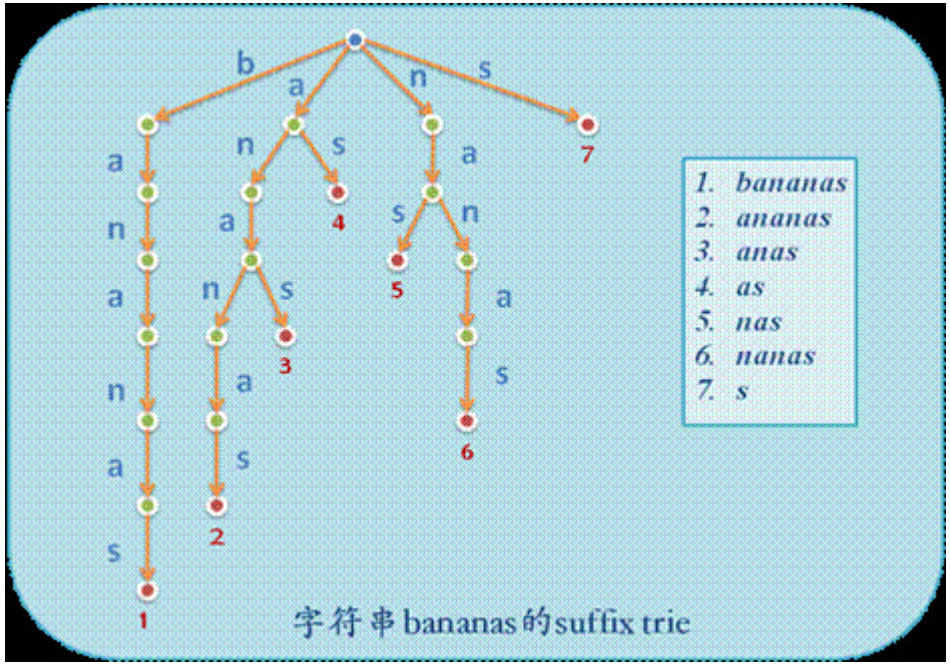
一个长度为 n 的字符串 S，它的后缀树定义为一棵满足如下条件的树：
1. 从根到树叶的路径与 S 的后缀一一对应。即每条路径惟一代表了 S 的一个后缀；
2. 每条边都代表一个非空的字符串；
3. 所有内部节点（根节点除外）都有至少两个子节点。性质：长度为 n 的字符串的后缀树的节点个数为 O(n)。



后缀树

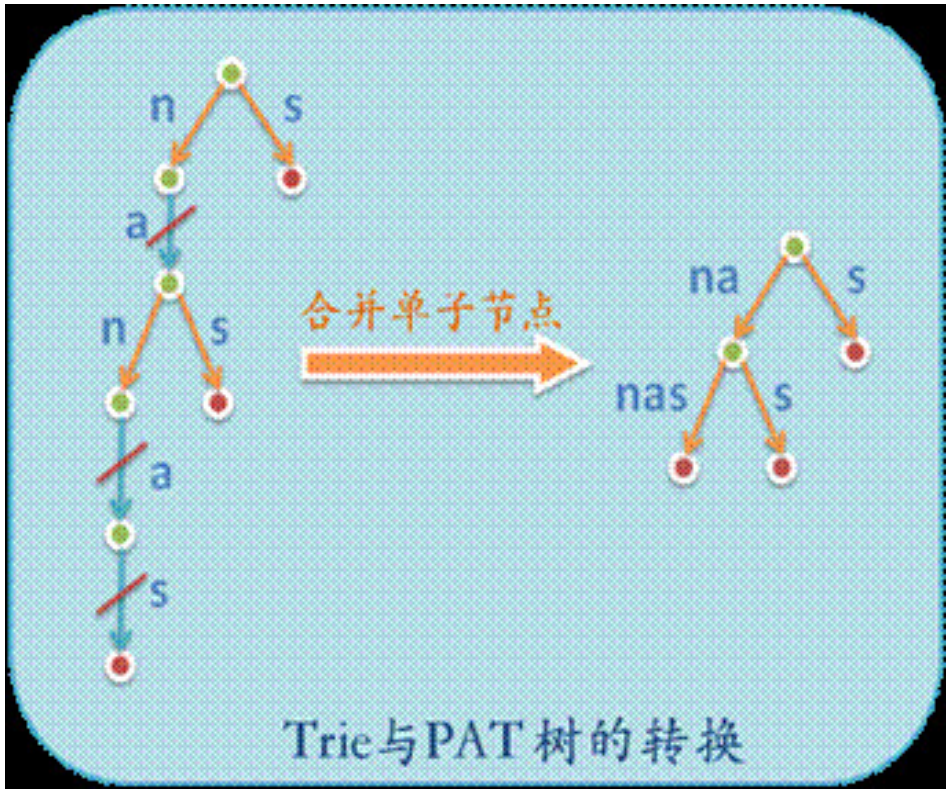
暴力算法

要构建一个字符串的后缀树，只需要将其所有的后缀建成一棵 trie 树，然后压缩所有只有一个儿子的点。



后缀树

暴力算法



后缀树

应用

虽然现在暴力建立后缀树的复杂度还无法接受，但是让我们把它先放在一边，体会一下它的强大。

- 求两子串最长公共前缀。找到这两个子串对应的后缀对应的叶子，求两叶子的 LCA。
- 求解一个字符串有多少个不同子串。只需构建后缀树，然后计算每个节点对应的子串个数之和即可。
- 给定一个字符串，对 i=1...n，求出现次数最多的长度为 i 的子串的出现次数。后缀树上一个节点代表的子串的出现次数等于它子树中叶子个数。枚举后缀树的每个节点求解即可。
- 给定一个字符串，出现次数至少为 K 次的最长子串有多长。同理（后缀数组经典）

后缀树

应用

- 给定一个字符串，出现次数至少两次的最长子串有多长。这里的出现两次不能重叠。ababa 中 aba 出现一次，ab 出现两次。给每个叶子用它是第几个后缀编号。记录每个节点子树中编号最小值和最大值，二分答案，检查每个节点中有没有编号相差大于等于当前二分值的两个叶子。(后缀数组经典)
- 求两字符串的最长公共子串。构建字符串 s#t，建立后缀树，给每个叶子染黑白染色，枚举每个节点看它子树中是不是同时存在黑白叶子，更新答案。

后缀树

后缀树的快速构建

希望从后往前来增量构建后缀树。因为这样只需在后缀树上一次增加一条链，然后在原本的后缀树上做少量修改即可。考虑添加一条链的情况。这时需要快速找到新添加的这条链最多可以走到哪个节点，然后在这个节点下创建儿子。

各种情况

添加一个字符时，可能出现三种情况。

1. 有可能还没有添加过这个字符，那直接在根节点下新增一个儿子即可。
2. 有可能匹配到的最长字符串正好是某个节点代表的最长字符串。这时原树的节点不需要分裂，只需要在这个节点下新增一个儿子即可。
3. 有可能匹配到的最长字符串 s' 并非某个节点代表的最长字符串，而是某节点代表的中间字符串。这时原节点分裂，原节点中比 s' 长的字符串构成另一个节点，成为它的儿子。同时再新增一个儿子表示新增的字符串。(如 babc)

后缀树

重要引理

发现快速找到匹配到的最长字符串所在的节点即可。
从长到短枚举当前字符串的前缀，尝试在这个前缀前面加一个字符，看看得到的字符串是否被当前的后缀树包含。第一个找到的就是结果。
引理：后缀树中同一个节点代表的所有字符串在前面加同一个字符，得到的这些字符串也在同一个节点中（如果被后缀树包含的话）。
证明：反证法，如果添加的字符是 a，且 s1,s2 在同一个节点，as1,as2 不在同一个节点。
那么以 a 所在节点为根的子树实际上是把字符串中所有 a 开头的后缀加入 trie 树压缩的结果。
把这些 a 开头的后缀的第一个字符去掉，也是原字符串的后缀。那么，as1 和 as2 分裂了，s1 和 s2 当然也会分裂，产生了矛盾。

后缀树

快速维护

于是，我们需要维护一个辅助数组：后缀树中每个节点代表的字符串前面加一个字符可以转移到哪个节点。trans[N][26]
于是，在增量构建后缀树时，我们需要同时维护两个东西。一个是后缀树上节点的父亲，另一个是每个节点的转移边。
考虑转移边的维护。每次在字符串的前面新增一个字符时，能够被影响转移边的节点一定是原字符串的前缀。也就是代表整个字符串的节点和它后缀树上的所有祖先。
那么我们不断跳父亲，如果当前节点没有这个字符对应的转移边，那么就简单地把这个节点对应字符的转移边置为这个将要新建的节点即可。

后缀树

快速维护

直到找到了一个节点 p，它有这个字符对应的转移边。（跳到根还找不到就万事大吉，是情况 1）
此时找到了匹配到的最长字符串所在的节点 q。查看匹配到的字符串是不是 q 的最长字符串。如果是，则是情况 2，简单地在这个节点下新增一个儿子即可。
否则，是情况 3，需要执行节点分裂操作，将 q 分裂为包含较短字符串的 nq 和包含较长字符串的 q。此时需要继续跳父亲，把终点是 q 的转移边都改为终点为 nq。
之后处理新增节点 nq 的信息。nq 的父亲显然为 q。nq 的转移边与 q 的转移边完全相同。这基本也是因为上面那个引理。分裂处理完之后，简单地在 q 节点下新增一个儿子即可。

目录

后缀自动机

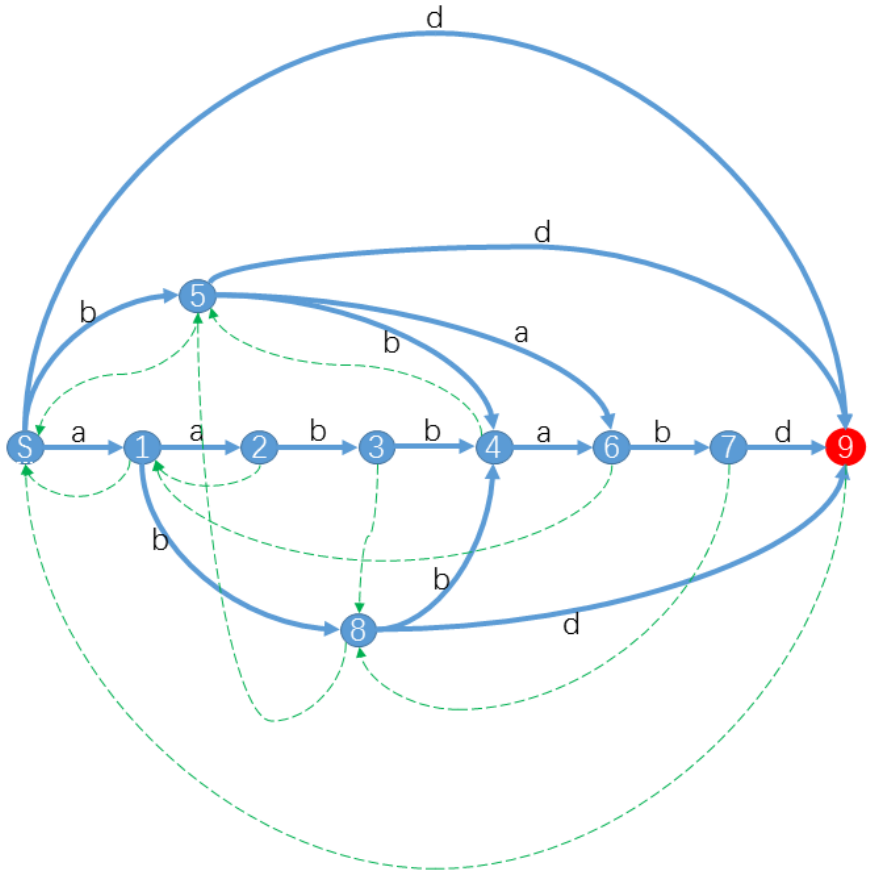
后缀自动机的定义

后缀自动机与 AC 自动机类似，也是一种自动机。它的状态集合是给定字符串的所有子串的集合。
对于每个状态，如果在它代表的字符串末尾添加一个字符可以得到字符串的另一个子串，则有一个转移边。
然而，由于这样的自动机状态数太多，是本质不同子串个数，$O(n^2)$ 级别。于是，需要通过某种方式压缩状态量。规定一个子串的 endpos 集合是它在原串中出现位置的集合。规定 endpos 集合完全一样的所有子串构成同一个状态。
经过这样的压缩，状态量成功达到 $O(n)$ 级别。
一个节点包含的子串为从起始节点到这个节点的所有合法路径。
同时，后缀自动机还有一个附属结构，除起始状态外，每个状态都有一个 link 指针，指向状态中包含的字符串的最长的不在此状态中的后缀所在的状态。link 指针构成了一棵树。

后缀自动机

图

这是 aabbabd 的后缀自动机。



后缀自动机

后缀自动机与后缀树

实际上，我们上面介绍的使用 trans[N][26] 辅助构造后缀树的算法就是构造后缀自动机的算法。在之前提到的算法中，我们从后往前添加字符。实际上，只需要改成从前往后添加字符，就得到了后缀自动机的构造算法。经过构造算法得到的所有的点就是后缀自动机的状态集合，trans[N][26] 就是后缀自动机的转移函数。注意到我们原来对 trans[N][26] 的定义就是在字符串的前面加一个字符转移到的节点。现在翻转，当然就是在字符串的后面添加字符转移到的状态，满足后缀自动机的定义。

而在构造后缀自动机时我们可以顺便得到一棵 parent 树（或称 link 树等），它其实就是反串的后缀树。

后缀自动机

代码

```
void extend(int c){
    int cur=++tot,p;cnt[cur]=1;
    mxlen[cur]=mxlen[lst]+1;
    for(p=lst;p&&!trans[p][c];p=link[p])trans[p][c]=cur;
    if(!p)link[cur]=1;
    else{
        int q=trans[p][c];
        if(mxlen[q]==mxlen[p]+1)link[cur]=q;
        else{
            int clone=++tot;
            memcpy(trans[clone],trans[q],sizeof(trans[q]));
            mxlen[clone]=mxlen[p]+1;link[clone]=link[q];
            for(;p&&trans[p][c]==q;p=link[p])trans[p][c]=clone;
            link[cur]=link[q]=clone;
        }
    }
    lst=cur;
}
int main(){
    scanf("%s",s+1);n=strlen(s+1);lst=1;tot=1;
    for(int i=1;i<=n;i++)extend(s[i]-'a');
```

后缀自动机

后缀自动机的性质

- SAM 是一个 DAG。
- SAM 的节点数不超过 2n-1。上述构造算法就是证明。（后缀自动机题目需要把空间开二倍）
- SAM 的边数不超过 3n-4。证明略。这点基本保证了上述构造算法的时间复杂度。因为在后缀树上跳父亲的次数是和 SAM 边数相关的。（基本每跳一次就建立一条新边）
- SAM 中每个节点包含的字符串是它包含的最长子串的一些连续长度的后缀。
- 从自动机的起始节点到自动机上任意一个节点的任意一条合法路径都对应了原串的一个子串。

目录

后缀数组

后缀数组简述

后缀数组算法可以 $O(n)$ 或 $O(n \log n)$ 求出一些关键的数组。
rk[i] 表示第 i 个后缀在所有后缀中的排名（按字典序排序）
sa[i] 表示排序后第 i 个后缀在原串中的位置（rk[i] 的逆）
height[i] 表示排序后第 i 个后缀与第 i-1 个后缀的最长公共前缀长度。
通常，还会在 height 数组上做 ST 表，以快速求 height 数组一段区间的最小值。

后缀数组

后缀数组与后缀树

这些数组在后缀树上都有对应。

sa[i] 其实就是在后缀树上按字符顺序 dfs，依次 dfs 到的叶子的编号顺序。

height[i] 其实就是 dfs 序排序的第 i 个叶子和第 i-1 个叶子的 LCA 深度。(未压缩时)

后缀数组求两后缀的最长公共前缀时，首先查 rk，设两个 rk 为 l 和 r。然后 ST 表求 height[l+1] 到 height[r] 的最小值。这实际上就是在求后缀树上两个叶子的 LCA。

后缀数组其余的问题也可以使用后缀树的想法来思考，可能会柳暗花明。

后缀算法的比较

后缀算法的比较

由于后缀树的性质太好了，实际上至少有百分之 80 的后缀自动机的题目本质上都只是利用后缀自动机建立了一棵后缀树，之后利用后缀树的性质解决问题。我们把后缀自动机和后缀数组与后缀树比较一下。
我们可以发现，使用后缀树可以简单地 O(n) 构造后缀数组。而使用后缀数组，我们可以通过与构造虚树类似的方法来 O(n) 构造后缀树。
而后缀自动机是在后缀树的基础上添加了后缀树没有的结构，即 trans 转移边。
所以，我在这里下一个暴论：
后缀自动机 $>$ 后缀树 $\approx$ 后缀数组。
当然后缀自动机也有它的基本上唯一的缺点：需要的空间大。它需要开 trans[N][26] 这个数组，于是空间天然乘 26。少数情况卡空间时可能还是只能用后缀数组。

目录

广义 SAM

Description

广义 SAM，一般解决多个字符串的相关问题。它基本就是在很多个串上建立的 SAM。
给定一个字符串集合，广义 SAM 就是一个包含这个字符串集合中所有子串为状态的自动机。
相应地，广义 SAM 的 parent 树就对应着将集合中所有字符串的反串的所有后缀插入 trie 树，再压缩得到的“广义后缀树”。

构建

建立广义 SAM，最正确的写法是先建 trie 树，然后在 trie 树上 bfs，一个一个在 SAM 中添加字符来增量构建。而 SAM 的核心代码逻辑基本上是没有变化的。
广义 SAM 的构建有众多流行且错误的写法，具体可以查看这篇博客：
https://www.luogu.com.cn/blog/ChenXingLing/solution-p6139

广义 SAM

性质

若字符串集合的总长为 n，则广义 SAM 的节点数、边数关于 n 仍然是线性的。这保证了它的复杂度。
对字符串集合构建广义 SAM，总长为 n，集合中每个字符串的子串所在的状态数之和是 $O(n\sqrt{n})$ 级别的。这是因为长度大于 $\sqrt{n}$ 的字符串不会超过 $\sqrt{n}$ 个，每个字符串最多 $O(n)$ 个状态，是 $O(n\sqrt{n})$。长度为 $i(i < \sqrt{n})$ 的字符串的子串所在状态数不会超过 i^2 个，而 $\sum i \leq n$，则 $\sum i^2 \leq \sqrt{n} \sum i \leq n\sqrt{n}$。

题目

给定一个字符串集合。对集合中每个字符串，询问它的所有本质不同子串在字符串集合中出现次数之和。一个子串是集合中某个字符串的子串算作出现一次，在不同字符串中出现、在同一字符串不同位置出现均算作不同的出现。

广义 SAM

解答

首先构建广义 SAM。初始将每个字符串的每个前缀所在的节点 val 值加一，代表它出现了一次。然后在 parent 树上做子树和，因为一个前缀的所有后缀都出现了一次，即它 parent 树上到根的链上所有状态都出现了一次。每次直接找出要查询的字符串 s 的所有本质不同子串在 SAM 中所在的节点，把这些节点的 val 值乘以它代表的子串个数都累加加起来即可。s 的所有本质不同子串在 SAM 中所在的节点可以通过直接打标记暴力跳父亲来不重不漏地访问到。

```
for(int i=1;i<=n;i++)
    for(int j=1,p=1;j<=len[i];j++){
        p=trans[p][s[i][j]-'a'];
        for(int l=p;l;l=par[l])
        if(flag[l]!=i){
            //do something
            flag[l]=i;
        }
        else break;
```

广义 SAM 与 AC 自动机

广义 SAM 与 AC 自动机

广义 SAM 和 AC 自动机都是处理多串问题的利器。广义 SAM 毋庸置疑的可以解决很多 AC 自动机无法解决的问题。但是，一般 AC 自动机能够解决的问题也可以使用广义 SAM 来解决。可以说，广义 SAM 在一定程度上是 AC 自动机的上位替代。当然，这里没有考虑代码难度等问题。关于如何使用广义 SAM 解决 AC 自动机相关的问题，可以查看这篇博客：
https://prutekoi.github.io/post/guan-yu-ac-zi-dong-ji-he-guang-yi-sam/

目录

回文自动机

Description

回文自动机是解决回文相关问题的利器。
一个字符串的回文自动机其实是两棵 trie 树。一棵存放原字符串的奇回文子串，另一棵存放原字符串的偶回文子串。
一个字符串能够被奇 trie 树表示出来当且仅当它是原字符串的奇回文子串的一半。若原串有子串 abcba，则 cba 在奇 trie 树中。偶 trie 树也同理。
除 trie 树外，回文自动机还维护一个关键的辅助数组。trie 树上每个节点都有一个失配指针 link，类似 kmp 的 next，AC 自动机的 fail，link 表示当前节点代表的回文串的最长回文后缀所在的节点。link 指针构成了一棵树。

回文自动机

构建

回文自动机使用类似后缀自动机的增量构建。注意到，每次在一个字符串的末尾添加一个字符，最多增加一个本质不同的回文子串。这个回文子串删去头尾，一定是原字符串的一个回文后缀。所以，我们只需从长到短枚举原字符串的所有回文后缀，直到找到一个回文后缀 a，它的前一个字符和当前要添加的字符相同，则找到了一个新的回文子串 b，如果 trie 树上没有这个回文子串，则在节点 a 上添加一个儿子，表示这个新的回文子串。新回文子串的 link 指针则基本上把同样的事情再做一次即可。

回文自动机

代码

```
char s[N];int n,ans,lst,cnt,len[N],fail[N],ch[N][26],d[N];
void init(){fail[0]=fail[1]=cnt=1;len[1]=-1;}
void extend(int i){
    int c=s[i],p=lst;
    while(s[i-len[p]-1]!=s[i])p=fail[p];
    if(ch[p][c]){lst=ch[p][c];return;}
    int x=fail[p],y=++cnt;
    while(s[i-len[x]-1]!=s[i])x=fail[x];
    fail[y]=ch[x][c];len[y]=len[p]+2;
    lst=ch[p][c]=y;d[y]=d[fail[y]]+1;
}
int main(){
    scanf("%s",s+1);n=strlen(s+1);init();
    for(int i=1;i<=n;i++){
        extend(i);
    }
```

时间复杂度分析

每次在 while 中跳一次 link，lst 在 link 树上的深度就会减一。每次 extend 只会加一，所以深度最多减 n 次，即时间复杂度为 O(n)。

回文自动机

题目

求字符串中有多少个本质不同回文子串。回文自动机的节点数即为答案。

[APIO2014] 回文串

定义 s 的一个子串的存在值为这个子串在 s 中出现的次数乘以这个子串的长度。求回文子串存在值的最大值。

建立回文自动机。在建立的过程中把 s 每个前缀的最长回文后缀的 val 值加一。在 link 树上做子树和。然后枚举所有节点，求每个节点代表的回文串长度 *val 值的最大值即可。

目录

Thank you! Any Question?