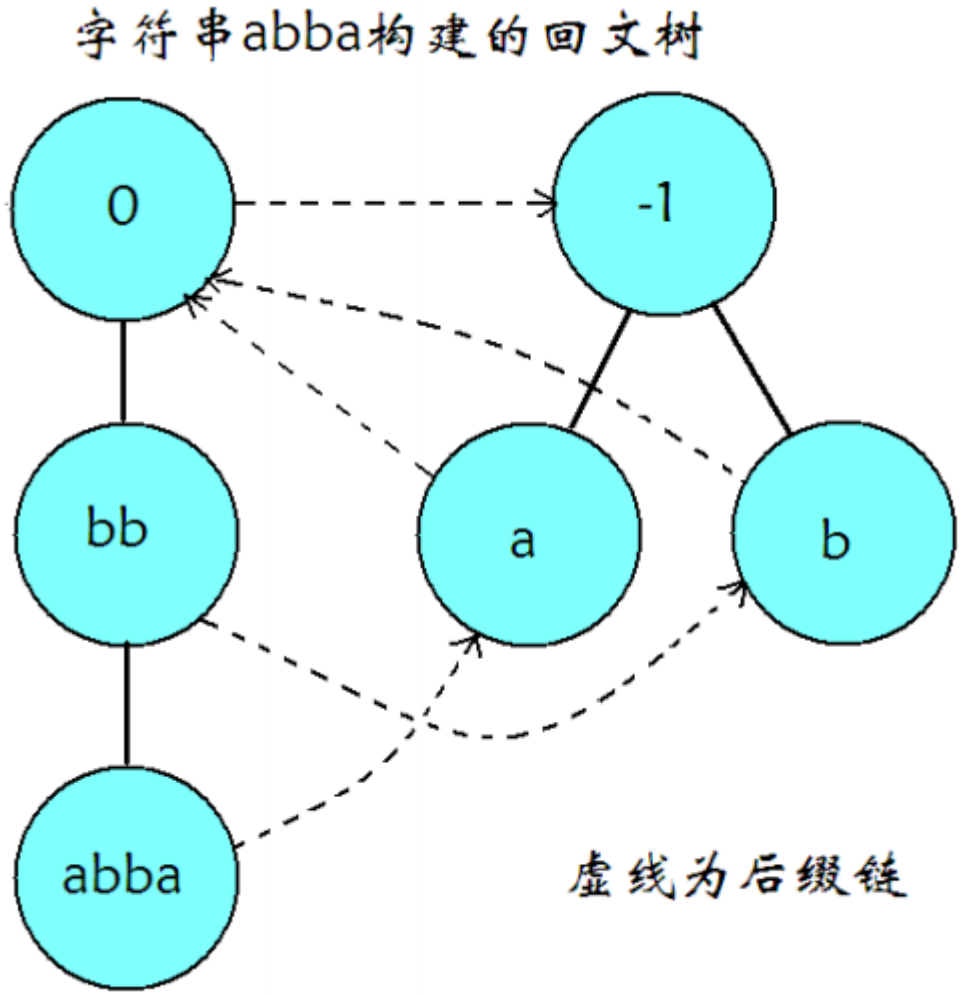
回文树

结构

回文树大概长这样：



功能

假设我们有一个串 S□S 下标从 0 开始，则回文树能做到如下几点：

1. 求串 S 前缀 $0\sim i$ 内本质不同回文串的个数（两个串长度不同或者长度相同且至少有一个字符不同便是本质不同）。
2. 求串 S 内每一个本质不同回文串出现的次数。
3. 求串 S 内回文串的个数（其实就是 1 和 2 结合起来）。
4. 求以下标 i 结尾的回文串的个数。

模板

```
const int MAXN = 100005 ;
const int N = 26 ;

struct Palindromic_Tree {
    //cnt最后count一下之后是那个节点代表的回文串出现的次数
    int next[MAXN][N] ;//next指针□next指针和字典树类似，指向的串为当前串两端加上同一个字符构成
    int fail[MAXN] ;//fail指针，失配后跳转到fail指针指向的节点
    int cnt[MAXN] ; //表示节点i表示的本质不同的串的个数（建树时求出的不是完全的，最后count()函数跑一遍以后才是正确的）
    int num[MAXN] ; //表示以节点i表示的最长回文串的最右端点为回文串结尾的回文串个数
    int len[MAXN] ;//len[i]表示节点i表示的回文串的长度（一个节点表示一个回文串）
    int S[MAXN] ;//存放添加的字符
    int last ;//指向新添加一个字母后所形成的最长回文串表示的节点。
    int n ;//表示添加的字符个数。
    int p ;//表示添加的节点个数。

    int newnode ( int l ) {//新建节点
        for ( int i = 0 ; i < N ; ++ i ) next[p][i] = 0 ;
        cnt[p] = 0 ;
        num[p] = 0 ;
        len[p] = l ;
        return p ++ ;
    }

    void init () {//初始化
        p = 0 ;
        newnode (  0 ) ;
        newnode ( -1 ) ;
        last = 0 ;
        n = 0 ;
        S[n] = -1 ;//开头放一个字符集中没有的字符，减少特判
        fail[0] = 1 ;
    }

    int get_fail ( int x ) {//和KMP一样，失配后找一个尽量最长的
        while ( S[n - len[x] - 1] != S[n] ) x = fail[x] ;
        return x ;
    }

    void add ( int c ) {
        c -= 'a' ;
        S[++ n] = c ;
        int cur = get_fail ( last ) ;//通过上一个回文串找这个回文串的匹配位置
        if ( !next[cur][c] ) {//如果这个回文串没有出现过，说明出现了一个新的本质不同的回文串
            int now = newnode ( len[cur] + 2 ) ;//新建节点
```

```
            fail[now] = next[get_fail ( fail[cur] )][c] ;//和AC自动机一样建立fail指针，以便失配后跳转
            next[cur][c] = now ;
            num[now] = num[fail[now]] + 1 ;
        }
        last = next[cur][c] ;
        cnt[last] ++ ;
    }

    void count () {
        for ( int i = p - 1 ; i >= 0 ; -- i ) cnt[fail[i]] += cnt[i] ;
        //父亲累加儿子的cnt，因为如果fail[v]=u，则u一定是v的子回文串！
    }
} ;
```