

Palindrome Tree小结

简介

- 回文树，作为一个2014年刚提出的复杂度仅为O(n)的高效算法，其功能强大且简单易懂，是处理回文串问题的一大利器。

结点&边

结点

- 本质上来说，回文树是一个自动机，该自动机的每个结点都是一个回文串。

如 (baab) (abaaba) (aba)

- 每个结点要是都存下完整的字符串会大量占用空间，故实际使用时只存长度。

边

- 回文树内置了两种边：

- 一种是类似Trie的指向儿子的边(son)；

(baab) =====> (abaaba) (A——>xAx)

- 一种是类似KMP的指向fail的边(suffix)。

(abaaba) ——> (aba) (指向除自己以外的最长的是回文串的后缀)

回文树的构造

注：下文以对字符串构造回文树做例子

根

- 回文树有两个树根，一个长度为-1，一个长度为0。
- 两个根的目的是方便同时处理长度为奇数与偶数的串。

插入字符

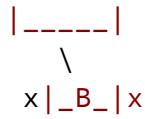
```
| <-----s-----> |
| <-----p-----> |
===== (字符串) ======x=====
 | -----t----- |
```

- 假设字符串s已经处理了p，正在处理x字符。
- t是p的最长回文串后缀。

```
| <-----s-----> |
| <-----p-----> |
===== (字符串) ======x=====
 | -----t----- |
    \ 
    | ----- |
       \ 
       x | __A__ | x
          \ 
          | ----- |
             \ 
             | --- |
```

- 首先不断走suffix边直到找到一个串A，它的前一个字符也是x。
- xAx 是字符串p的最长回文串后缀。
- 建立一条 A=====>xAx 的边(如果已有则return)。
- 现在构建 xAx 的suffix边。

```
| <-----s-----> |
| <-----p-----> |
===== (字符串) ======x=====
 | -----t----- |
    \ 
    | ----- |
       \ 
       x | __A__ | x
          \
```



- 从 xAx 继续出发直到走到 xBx (如果 $\text{len}(xAx)=1$, xBx 为空串)。
- 则 xBx 一定在回文树中(xBx 在 xAx 的前缀出现过)。
- 加一条 $xAx \rightarrow xBx$ 的suffix边。

代码

```

int totn,suff;
char s[maxn];
struct node{
    int len,son[26],suffix;
}T[maxn];

void build_tree()
{
    totn=suff=2;
    T[1].len=-1; T[1].suffix=1;
    T[2].len=0;  T[2].suffix=1;
}

void insert_tree(int id)
{
    int cur=suff,alphabet=s[id]-'a';
    for (;s[id-T[cur].len-1]!=s[id];cur=T[cur].suffix);
    if (T[cur].son[alphabet])
    {
        suff=T[cur].son[alphabet];
        return;
    }
    suff=T[cur].son[alphabet]=++totn;
    T[totn].len=T[cur].len+2;
    if (T[totn].len==1)
    {
        T[totn].suffix=2;
        return;
    }
    cur=T[cur].suffix;
    for (;s[id-T[cur].len-1]!=s[id];cur=T[cur].suffix);
    T[totn].suffix=T[cur].son[alphabet];
}

```

例題

- [spoj NUMOFPAL](#)
- [Ural 1960](#)
- [CodeForcesGym 100548G](#)