回文分解O(nlogn)算法的理论与实践

吴东学，杨鹏

(电子科技大学数学科学学院,成都 611731)

摘要

随着生物科学的发展与进步，关于模式处理的算法的重要形越发凸显。本文实现一个在线分解回文串的算法，并阐述它的复杂度是O(nlogn)。

另外本文仅仅是对[1]在相应章节上进行了更加细致的阐述。

关键字:回文分解　O(nlogn)

1　引言

回文分解问题是将一个字符串分解成最少的回文串的组合。这个算法在模式处理有一定的作用，且用到了比较前沿的数据结构eertree，能在O(nlogn)的复杂度内有效解决了该问题。

2　相关定义

回文串

给定字符串S[1,2,...,n]，若S[1,2,...,n]=S[n,n-1,...,1]，则称字符串S[1,2,...,n]为回文串。

回文分解

给定字符串S[1,2,...,n]，将其分解为最少的回文子串的组合。

3　暴力算法

采用动态规划的方法，我们假设$ans[n]$为字符串$S[1,2,...,n]$的最小回文分解数。若$S[i,i+1,...,n]$是字符串$S[1,2,...,n]$中的一个回文串，则对于所有的合法以n结尾的回文串$S[i,i+1,...,n]$，显然$ans[n]=min(ans[i-1]+1)$。

分析上述算法的复杂度，对于字符全部相同的字符串，计算ans[n]的复杂度是$O(n)$，因此整个算法的复杂度是$O(n^2)$。

现在考虑使用Eertree解决此问题。

4　Eertree

Eertree的构造

这是一种trie形数据结构，每个节点表示一个回文串，并存储该回文串的长度，每条边表示在上一个节点的首尾加上该边对应的字符。除此之外，在每个节点再维护一个后缀链接，指向该节点在Eertree中的最长后缀回文串节点。

```
struct Node
{
    int length;
    Node * next[26];
    Node * link;
}
Node * newnode(Node * pre,Node * sufflink)
{
    Node * cur = new Node;
    cur->length = pre->length+2;
    cur->link = sufflink;
    return cur;
}
```

我们初始化两个节点，一个长度为-1的节点，另一个长度为0的节点，并且把长度为0的节点的后缀链接指向长度为-1的节点。

考虑对其的增量法构造，我们需要维护当前字符串最长后缀回文节点，我们记为Maxsuff

现在新增字符a，那么此时我们需要找到aTa，作为新的最长后缀回文。对于T，它是当前的一个后缀回文，那么沿着Maxsuff的后缀链接往上查找，就一定可以找到T。

那么找到T之后，我们可能需要找到aTa的后缀链接，对于这个问题，显然只要继续沿着T的后缀链接往上查找，就可以解决。

```
Node * suffgo(Node * cur,char buf[],int pos)
{
    while(buf[pos - 1 - cur->length] != buf[pos])
        cur = cur->link;
    return cur;
}
void update(char buf[],int pos)
{
    Node * cur = suffgo(Maxsuff,buf,pos);
    int alp = buf[pos]-'a';//对于字符集合进行处理
    if(cur->next[alp] == NULL)
    {
        if(cur->length > 1)
            cur->next[alp] = newnode(cur,suffgo(cur->link,buf,pos)->
                next[alp]);
        else
            cur->next[alp] = newnode(cur,node of length 0); //链接在
                长度为0的节点上
    }
    Maxsuff=cur->next[alp];
}
```

Eertree的应用

1 给定一个字符串，询问该字符串中本质不同的回文串的数量

2 给定一个字符串，询问某个回文串是否存在

3 给定一个字符串，询问在以j位置结尾的回文串的数量

5 在Eertree的另一种后缀连接

定义新的Eertree节点

```
struct Node
{
    int length,diff,dp;
    Node * next[26];
    Node * link;
    Node * slink;
}
Node * newnode(Node * pre,Node * sufflink)
{
    Node * cur = new Node;
    cur->length = pre->length+2;
    cur->link = sufflink;
    cur->diff = cur->length - sufflink->length;
    if(cur->diff == cur->link->diff)
        cur->slink = cur->link->slink;
    else
        cur->slink = cur->link;
    return cur;
}
```

这样构造Eertree之后，我们可以每次用一定的技巧维护每次查询的值了。

6 算法与证明

首先一个回文串的最长后缀回文一定可以是该回文串的某个前缀。

考虑某一条后缀链接上diff值相同的部分，我们可以发现当前需要访问的ans值都在之前的后缀链接上访问过，由此我们可以采用动态规划方法，将diff值相同节点的答案合并在一起。

```
void trans(int n)
{
    ans[n] = inf;
    for(Node * v = suff ; v->length > 0 ; v = v->slink)
    {
        v->dp = ans[n-(v->slink->length + v->diff)];
        if(v->diff == v->link->diff)
            v->dp = min( v->dp , v->link->dp);
        ans[n] = min (ans[n] , v->dp);
    }
}
```

那么每次更新的代价就是沿着slink跳跃的复杂度。

引理 在一颗EErtree中，slink的路径长度为O(logn)[2]。

由此，我们得到了在O(nlogn)的时间范围内解决回文分解的算法。

References

[1] Mikhail Rubinchik and Arseny M.Shur:$EERTREE$: *An Efficient Data Structure for Processing Palindromes in Strings*

[2] Kosolobov,D.,Rubinchik,M.,Shur,A.M.:Pal^k *is linear recognizable online*