时间、空间复杂度分析

> problem2.pdf 对该程序的时间复杂度和空间复杂度进行分析。

```
//move函数时间复杂度O(n)
//空间复杂度O(1)
template <class elemType>
typename sLinkList<elemType>::node * sLinkList<elemType>::move(int i)
const
{
    node *p = head;
    //while循环时间复杂度最高，为O(n)
    while(i-- >= 0) p = p->next;
    return p;
}
```

```
//insert函数时间复杂度O(n)
//空间复杂度O(1)
template <class elemType>
void sLinkList<elemType>::insert(int i, const elemType &x)
{
    node *pos;
    //move函数时间复杂度最高，为O(n)
    pos = move(i - 1);
    pos->next = new node(x,pos->next);
    ++currentLength;
}
```

```
//remove函数时间复杂度O(n)
//空间复杂度O(1)
template <class elemType>
void sLinkList<elemType>::remove(int i)
{
    node *pos, *delp;
    //move函数时间复杂度最高，为O(n)
    pos = move(i - 1);
    delp = pos->next;
    pos->next = delp->next;// 绕过delp
    delete delp;
    --currentLength;
}
```

```
//getSuffix函数时间复杂度O(n2)
//空间复杂度O(n)
friend sLinkString<elemType> & getSuffix(const sLinkString<elemType> &
string1, const sLinkString<elemType> & string2, sLinkString<elemType> &
result)
    {
        //for循环：时间复杂度O(n2)，空间复杂度为O(n)
        for(int i = 1;
            string1.move(string1.length() - i)->data ==
string2.move(string2.length() - i)->data;//比较：时间复杂度O(n)
            ++i)
            //插入：空间复杂度O(1)
            result.insert(0, string1.move(string1.length() - i)->data);
        return result;
    }
```

```
//traverse函数时间复杂度O(n)
//空间复杂度O(1)
template <class elemType>
void sLinkList<elemType>::traverse() const
{
    node *p = head->next;
    //while循环时间复杂度为O(n)
    while (p != NULL) {
        cout << p->data << " ";
        p=p->next;
    }
    cout << endl;
}
```

```
//main函数时间复杂度为O(n2)
//空间复杂度O(n)
int main()
{
    sLinkString<char> myString1, myString2, myString3;
    int iniString;

    //循环：时间复杂度O(n2)，//空间复杂度O(n)
    do
    {
        iniString = cin.get();
        if(iniString >= 'a' && iniString <= 'z')
            myString1.insert(myString1.length(), iniString);
            //insert函数时间复杂度O(n)
            //空间复杂度O(1)
        else if(iniString >= 'A' && iniString <= 'Z')
```

```
            myString1.insert(myString1.length(), iniString - 'A' + 'a');
            //insert函数时间复杂度O(n)
            //空间复杂度O(1)
    }
    while(iniString != '\n');

    do
    {
        iniString = cin.get();
        if(iniString >= 'a' && iniString <= 'z')
myString2.insert(myString2.length(), iniString);
        else if(iniString >= 'A' && iniString <= 'Z')
myString2.insert(myString2.length(), iniString - 'A' + 'a');
    }
    while(iniString != '\n');

    //getSuffix函数时间复杂度O(n2)
    //空间复杂度O(n)
    getSuffix(myString1, myString2, myString3);

    //traverse函数时间复杂度O(n)
    //空间复杂度O(1)
    myString3.traverse();

    return 0;
}
```