第4章 字符串

4.1 假设以链表结构 LString(定义如下)作为串的存储结构，试编写判别给定串 S 是否具有对称性的算法，并要求算法的时间复杂度为O(StrLength(S))。

解
算法描述
结合链表翻转的算法，将字符串的后一半翻转，再用前一半字符串与翻转后的后一半字符串逐字符比较，从而判断原字符串是否具有对称性。
复杂度分析
该算法可分为三部分：
1.寻找字符串中间位置
2.翻转后一半字符串
3.比较前一半字符串与后一半字符串
每一部分的时间复杂度均为O(StrLength(S))，故时间复杂度为O(StrLength(S))，由于该算法中翻转操作在原字符串上进行，只引入了常数个变量，故空间复杂度为O(1)。

```
//算法代码如下
struct listNode{
  char data;//存放数据
  ListNode*next;//存放指向后继结点的指针
}
typedef ListNode*ListPtr;
struct LString{
  ListPtr head;//链表的表头指针
  int strLen;//串的长度
}
bool judgePalindromeString(LString S){
  if(S.strlen<=1)//特判，S长度小于等于1时一定对称
    return true;
  ListPtr p=S.head;
  int len=S.strLen;
  for(int i=0;i<len/2-1;i++)//寻找字符串的中间位置 （因为没有头结点故找中间位置时范围为[0,len/2-1))
    p=p->next;
  //翻转后一半字符串
  ListPtr q=p->next;
  p->next=NULL;//断链
  p=NULL;//p指空，便于后续翻转操作
  while(q){
    ListPtr tmp=q->next;
    q->next=p;
```

```
      p=q;
      q=tmp;
    }
    //完成翻转，此时p指向翻转后的后一半字符串的第一个字符
    ListPtr r=S.head->next;
    while(r&&p){//逐字符比较
      if(r->data!=p->data)//有一处不等则不对称
        return false;
      r=r->next;
      p=p->next;
    }
    return true;//每一处都相等（或有一个字符串多出一位），对称
  }
```

4.2 写出一个线性时间的算法，判断字符串 T 是否是另一个字符串 T′ 的循环旋转。例如 arc 和 car 是彼此的循环旋转。

解

算法描述

若T和T′长度不同，则T和T′必不可能是彼此的循环旋转

若T和T′长度相同，则考虑将T′延长为原来的两倍变为T′T′，在T′T′中利用kmp算法进行对模式T的匹配。若匹配成功则说明T和T′为彼此的循环旋转，反之则不是。

复杂度分析

该算法主要由字符串拼接和kmp模式匹配算法两部分组成，每一部分的时间复杂度均为线性复杂度，故算法总的时间复杂度也是线性的，符合题目要求。

```
//算法代码如下
bool judgeCyclicRotatedStrings(string T1,string T2){
  int l1=T1.size(),l2=T2.size();
  if(l1!=l2)//T1和T2长度不同，则T1和T2不可能是彼此的循环旋转
    return false;
  T2+=T2;
  l2+=l2;//延长T2，同步更新长度
  //预处理T1的next数组
  vector<int>next(l1,0);
  next[0]=-1;
  int j=0,k=-1;
  while(j<l1){
    while(k>=0&&T1[j]!=T1[k])//递归式寻找最长首尾匹配真子串
      k=next[k];
    k++,j++;
    if(j==l1)//已经到达T1末尾
      break;
    next[j]=k;
    if(T1[j]==T1[k])//考虑能否优化处理
      next[j]=next[k];
```

```
    }
    //kmp模式匹配，T1为模式，T2为目标串，由于l2=2*l1，故可以直接开始匹配
    j=0;
    int i=0;
    while(j<l1&&i<l2){
      if(j==-1||T2[i]==T1[j])
        i++,j++;
      else
        j=next[j];
    }
    if(j==l1)//匹配成功，说明是循环旋转
      return true;
    else//匹配失败
      return false;
  }
```

4.3 请证明教材中next数组(优化和非优化两种)算法正确性。

证明
考虑任意的一个字符串P(0,...,n-1)对应的next数组
假设其作为模式与目标字符串T进行匹配（T足够长）

next[i]的含义
当P在P[i]处发生失配时（此时有1<=i<n且T[j]!=P[i]）下一次比较的位置（即下一次对P[next[i]]和T[j]进行比较）

下证next[i]即为P(0,...i-1)中最长首尾匹配真子串的长度.
根据匹配的规则易知当P在P[i]处发生失配时有P(0,1,...i-1)=T(j-i,j-i+1...,j-1).
若存在一个k，使得P(0,...,k-1)=P(i-k,...,i-1).
则将模式串向右滑动i-k位后P(0,...,k-1)必然与T(j-k,...,j-1)成功匹配，因此只需要直接比较P[k]与T[j]和它们之后的字符即可，即下一次匹配的位置next[i]=k.考虑到不能跳过可能的匹配位置，我们寻找一个最大的k让模式串的滑动距离i-k尽量的小即可.
因此next[i]=k即为P(0,...i-1)中最长首尾匹配真子串的长度.

下证next数组求法（非优化）的正确性
首先确定边界条件，令next[0]=-1;
若对模式的某一位置j-1已知next[j-1]=k，说明模式串P在P(0,...,j-2)中的最长首尾匹配真子串为P(0,...,k-1)=P(j-k-1,...,j-2)
我们要求模式串P在P(0,...,j-1)中的最长首尾匹配真子串
当P[k]=P[j-1]时，由于P(0,...,k-1)=P(j-k-1,...,j-2)，故P(0,...,k)=P(j-k-1,...,j-1)为P(0,...,j-1)中的最长首尾匹配真子串，长度为k+1，故next[j]=k+1=next[j-1]+1;
当P[k]!=P[j]时，P(0,...,k)!=P(j-k-1,...,j-1)此时可以将P(j-k-1,...,j-1)看作目标串，将P(0,...,k)看作模式串，它们在P[k]处发生了失配，根据next数组的含义，我们令k=next[k]，之后继续

进行匹配看能否匹配成功.循环此过程，直到匹配结束.
若成功寻找到k>=0，P(0,...,k)=P(j-k-1,...,j-1)，则next[j]=k+1;
当k=-1时说明P(0,...,j-1)的最长首尾匹配真子串为空串（即不存在），此时只能从头开始匹配，即next[j]=k+1=-1+1=0;
综上，next数组的非优化求法是正确的

下面补充说明next数组求法（优化）的正确性
我们考虑成功寻找到首尾匹配真子串的情况，即存在k>=0，next[j]=k.根据next数组的含义，当在j处发生失配时，要将下次匹配的位置移动到k，即下一次比较T[i]和P[j]
若P[j]=P[k]，有T[i]!=P[k]，必然不可能在P[k]处成功匹配，故可以继续移动比较位置，让next[j]=next[k].
综上，next数组的优化求法是正确的