使用动态规划的思想求解最大回文子串问题

原创 张佳辉 LOA算法学习笔记 2021-03-05 17:40

01 问题描述

给定一个字符串，输出字符串中的最长回文子串。回文串是指正反写都相同的字串。如输入"ababad"，最大回文子串为"ababa"。如果直接采用暴力求解，即枚举所有子串，再对每个子串进行判断，则需要 $\mathcal{O}(n^2)\cdot\mathcal{O}(n)=\mathcal{O}(n^3)$ 的复杂度。而通过动态规划的思想，可以将复杂度降至 $O(n^2)$ 。

02 动态规划算法

2.1 状态转移方程

观察可以发现，回文串是左右对称的结构，即首尾一定相同，去掉旧的首尾，新的首尾也相同。因此假设 $opt(i,j)$ 表示从i到j的子串是否为回文串，因此状态方程表示为：

$$\operatorname{opt}(i, j)=\begin{cases}0, & A(i) \neq A(j) \\ \operatorname{opt}(i+1, j-1), & A(i)=A(j)\end{cases}$$

单独的一个字符构成回文串，因此边界条件为 $opt(i,j)=\mathbf{true},(i=j)$ ，同时需要限定 $i\leq j$ 。因此我们只需要把这张表格的灰色区域填满，最后回溯就可以得到正确解。

	a	b	a	b	a	d
a	T					
b		T				
a			T			
b				T		
a					T	
d						T

2.2 注意状态转移顺序

确定状态转移顺序的时候需要注意，大子串的状态会用到小子串的状态值，如 $opt(0,4)$ (即"ababa")会转移到 $opt(1,3)$ (即"bab")，这时候需要规定填表顺序，保证的下一个转移到的子串的状态是确定的。

	a	b	a	b	a	d
a	T					
b		T				
a			T			
b				T		
a					T	
d						T

本文采用先确定小子串再确定大子串的顺序，即控制子串长由小到大，逐个循环填表。关键代码如下。循环之前需要单独处理长为2和1的子串。

```
//将长为1的子串填满为true
```

```
    for (int i=0; i<lenth; i++) {
      opt[i][i] = true;
    }
    //处理长为2的子串
    for (int i=0; i<lenth-1; i++) {
      if(s[i]==s[i+1])  opt[i][i+1] = true;
    }
    //从长度为3开始处理其它子串
    for (int k=3; k<lenth+1; k++) {
      for (int i=0; i<=lenth-k; i++) {
        opt[i][i+k-1] = (s[i]==s[i+k-1]) && opt[i+1][i+k-2];
      }
    }
```

执行过程中状态变化如下：

	a	b	a	b	a	d
a	T	F				
b		T	F			
a			T	F		
b				T	F	
a					T	F
d						T

	a	b	a	b	a	d
a	T	F	T			
b		T	F	T		
a			T	F	T	
b				T	F	F
a					T	F
d						T

	a	b	a	b	a	d
a	T	F	T	F	T	F
b		T	F	T	F	F
a			T	F	T	F
b				T	F	F
a					T	F
d						T

2.3 回溯求出最终解

从表格右上角开始沿右下对角线回溯，找到的第一个**true**即为最长回文子串所在的位置。

```
string find(string s) {
    int lenth = s.size();
    int l = -1;
    int r = -1;
    for (int k=lenth; k>=0; k--) {
      for (int i=0; i<=lenth-k; i++) {
        if ( opt[i][i+k-1] ) {
          l = i;
          r = i+k-1;
```

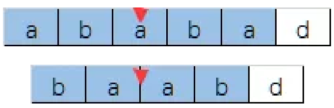
```
          break;
        }
      }
      if (l != -1) {
        break;
      }
    }
    string ans = s.substr(l,r-l+1);
    return ans;
}
```

2.4 复杂度分析

需要填一张 $n*n$ 的表格，因此时间复杂度和空间复杂度均为 $\mathcal{O}(n^2)$ 。

03 中心拓展法

上面的动态规划是从两边向中间转移，也可以利用对称的性质从中心向两边尽可能地拓展。这里需要注意，根据长度的奇偶性，最大子串的中心可能是一个元素，也可能在两个元素之间，如"ababa"的中心为'a'，"baab"的中心在两个'a'之间。



因此在遍历时需要同时考虑两种情况，通过中心点两边的坐标控制其位置。关键代码如下

```
//遍历每个可能的中心点
for (int i=1; i<lenth; i++) {
    oddNum = spread(i, i, s);//中心是一个元素
    evenNum = spread(i, i+1, s);//中心在两个元素之间
    int R = max(oddNum, evenNum);
    if ( R>maxR) {
      maxR = R;
      center = i;
    }
}
ans = s.substr(center-(maxR-1)/2, maxR);
//中心拓展函数
int spread(int l, int r, string s) {
  int lenth = s.size();
  while( l>=0 && r<lenth) {//左右元素相等，则向两边拓展
    if ( s[l]!=s[r] ) {
      break;
    }
    l--;
    r++;
  }
  return r-l-1;//返回当前中心的最大回文串
```

```
}
```

- **算法复杂度分析**

需要遍历 $2n$ 个中心点，每个中心点拓展需要 $\mathcal{O}(n)$ 的时间复杂度，因此总的时间复杂度为 $\mathcal{O}(n^2)$ ，空间复杂度只需要常数保存临时变量，因此总空间复杂度为 $\mathcal{O}(1)$ 。

04 总结

使用动态规划求解最大回文串的问题，关键是要注意填表的顺序，保证下一个状态是有效的，大家感兴趣也可以尝试其他的填表顺序。另外，还有一个方法是Manacher算法，它的思想基础是中心拓展法，但是时间复杂度可以达到 $\mathcal{O}(n)$ ，专门求解最大回文子串问题，大家有兴趣可以自行了解学习。

最后感谢您的阅读，因为水平有限，如有错误，欢迎指出~