

TI WAI HUA



MANACHAR 算法

CZC
2023-06-07
v2.1

算法背景

马拉车算法 Manacher 's Algorithm 是用来查找一个字符串的**最长回文子串**的**线性方法**，由一个叫 Manacher 的人在1975年发明的，这个方法的最大贡献是在于将时间复杂度提升到了线性。

manacher 可能是他↓



Department of Mathematics,
Statistics, and Computer Science
COLLEGE OF LIBERAL ARTS & SCIENCES

SUPPLEMENTAL PUBLIC UTILITIES

people

faculty
emeritus faculty
visiting scholars
teaching assistants
graduate students
associates
staff

seminars
course info
faculty research
chicago symposium

intranet

give to MSCS

update this page
print office sign

Glenn K. Manacher

Assoc. Professor Emeritus

Office: SEO 615
Phone: 312-413-2159
Email: manacher@uic.edu

Research Interests:

Computer algorithms; computer language design



算法基本信息

解决问题：求字符串中的回文串

预处理：因为处理回文串要分奇偶长度分类讨论
干脆一步到位，在串之间和首尾插入#号（回文中不会出现的字符）
这样长度为N的字符串，就变成了 $N+N+1$ 的串，
即 $2N+1$ 。 全部处理为奇数长度。

aaabbcc => #a#a#a#b#b#c#c#

Q这样处理影响回文性质么？

辅助数组LEN 重要性质

`len[i]-1` 就是以 `T[i]` 为中心的回文子串在原字符串中的长度。

例如：下图的a, $\text{len}[\text{pos}[\text{a}]] - 1 = 4 - 1 = 3$

T[i]	#	a	#	a	#	a	#	b	#	a	#
len[i]	1	2	3	4	3	2	1	4	1	2	1

辅助数组LEN 重要性质

$\text{len}[i]-1$ 就是以 $T[i]$ 为中心的回文子串在原字符串中的长度。

证明?

证明:

- 在转换后的字符串 T 中，显然回文字符串的长度为奇数。
- 假设已经求出 len 数组，则 $T[i]$ 的最长回文字符串的长度为 $2*\text{len}[i]-1$
- 记 $T[i]$ 所在的最长回文字符串长度为 N_i ，加入的分隔符 $\#$ 的个数是 N_i+1 ，总长度 N_i+N_i+1
- 有公式: $N_i+N_i+1 = 2*\text{len}[i] - 1$
- 所以 $N_i = \text{len}[i]-1$

求LEN数组

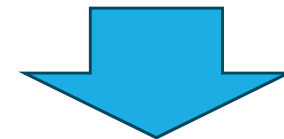
现在问题转化为了求len数组。

1. mx 为之前找到过的最靠右的回文字符串子串最右的位置
2. mid 为这个串的中间位置（加了#后的奇数串）
3. i 为当前所求的位置
4. j 为 i 关于 mid 的对称点

假设你有一个字符串

ABCSADWFAWSDAWASDFWFCAFW

内容是什么不重要，抽象成线



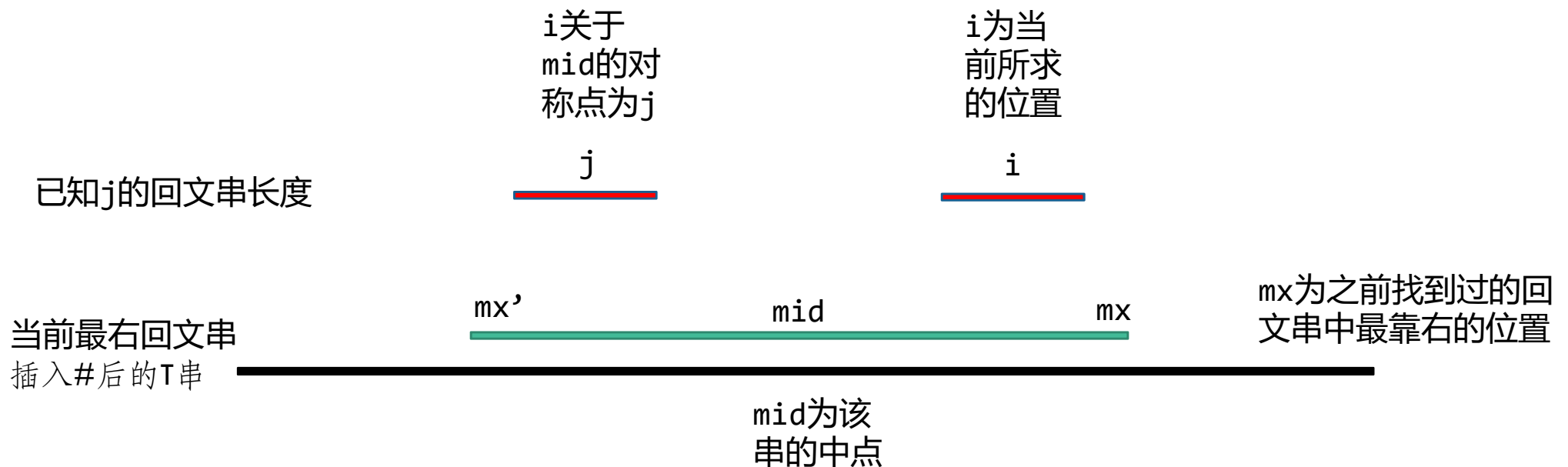
mx' j mid i mx

Case1 当 $i \leq mx$ 时

Case1.1 且当 $T[j]$ 回文串在 $[mx', mx]$ 内时(如图)

显然 $len[i] = len[j]$
因为回文串的
对称串也是回文串

更严格的证明:
采取反证法
假定答案 $len[i]$ 比当前更长?



Case1 当 $i \leq mx$ 时

Case1.1 且当 $T[j]$ 回文串在 $[mx', mx]$ 内时 显然 $len[i] = len[j]$

Case1.2 且当 $T[j]$ 回文串超过左边界 mx' 时(如图) $len[i] = mx - i + 1$

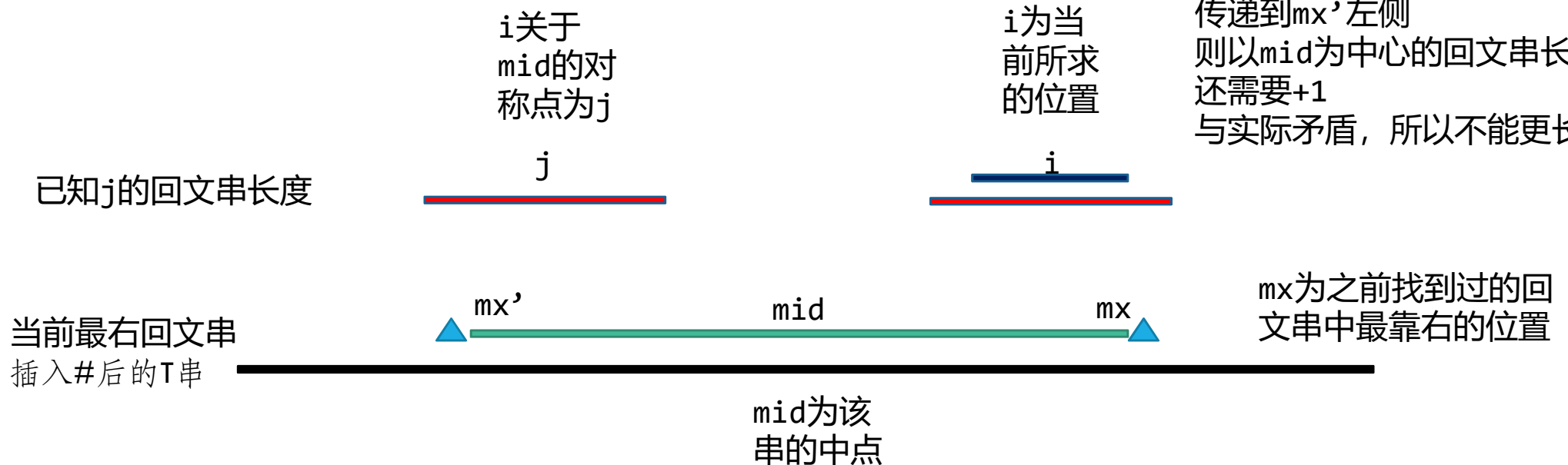
反证法可证明其正确性:

假定 i 中心的回文串还可更长
则需包括蓝色三角

由于 j 中心点的回文对称性
传递到 mx' 左侧

则以 mid 为中心的回文串长度
还需要+1

与实际矛盾, 所以不能更长。



Case1 当 $i \leq mx$ 时 $len[i] = len[j]$
 $len[i] = mx - i + 1$



$len[i] = \min(len[j], mx - i + 1)$

实际上当 $len[i] == mx - i + 1 == len[j]$ 的时候
没办法借助当前信息推测 i 中心回文串左右两端数据
还需要进一步处理

当 $len[j] == mx - i + 1$ 时
需要暴力判断2端点(▲)

Case1 当 $i \leq mx$ 时

$len[i] = \min(len[j], mx - i)$
再进行朴素算法检验是否可继续拓展

i 关于
 mid 的对
称点为 j

j

i 为当
前所求
的位置

i

已知 j 的回文串长度

mx'

mid

?

mx

当前最右回文串
插入#后的T串

mx 为之前找到过的回
文串中最靠右的位置

mid 为该
串的中点

Case2 当 $i > mx$ 时

朴素算。

伪代码:

```
FOR i in 1~n:  
    IF i<=mx:  
        len[i] =min(len[mid*2-i],mx-i)  
        朴素检验len[i]是否还能拓展  
    IF 找到更右的回文串:  
        更新mid,mx
```

易错点:

数组越界: 算到开头到0了还算。

解决方法: 在0位置添加一个特特殊的字符, 匹配失败停止

空间开大一点, 避免开02后不必要的错误。比如 $2N+100$

时间复杂度分析

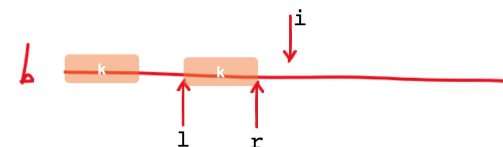
```
FOR i in 1~n:  
  IF i<=mx:  
    len[i] =min(len[mid*2-i],mx-i)  
    朴素检验len[i]是否还能拓展  
  IF 找到更右的回文串:  
    更新mid,mx
```

$O(n)$

2021年12月的关于exKMP讲解 Z函数-优雅方法

根据之前的信息，减少不必要的计算！
寻找限制条件下的状态转移函数，使得可以借助之前的状态来加速计算新的状态。
已经求出 $Z[1] \sim Z[i-1]$ ，正在求 $Z[i]$

记 $\max(Z[1], Z[2], \dots, Z[i-1])$
在 $i=1$ 时拿到最大值，其右端点记为 r



得益于mx信息的记录
算法只有遇到未匹配点时
才会在 $O(1)$ 的复杂度内完成匹配

代码实现

```
FOR i in 1~n:  
    IF i<=mx:  
        len[i] =min(len[mid*2-i],mx-i)  
    朴素检验len[i]是否还能拓展  
    IF 找到更右的回文串:  
        更新mid,mx
```

```
int manachar(string s) {  
    int mx = 0; 最右侧边界  
    int pos = 0; mx对应的对称中心  
    int ans = -1; 答案  
    for (int ptr = 1; ptr < s.size() - 1; ++ptr) {  
        if (ptr < mx)  
            lens[ptr] = min(lens[2 * pos - ptr], mx - ptr); → 当lens[对称点] > mx时,说明应当从mx处开始扩展  
        else  
            lens[ptr] = 1; 关于pos对称的点  
        while (s[ptr - lens[ptr]] == s[ptr + lens[ptr]]) 当ptr > mx时,扩展mx的边界  
            lens[ptr]++;  
        if (mx < ptr + lens[ptr]) { 扩展边界与对称中心  
            mx = ptr + lens[ptr];  
            pos = ptr;  
        }  
        ans = max(ans, lens[ptr] - 1); 当前位置为中心的最大回文串长度  
    }  
    return ans;  
}
```

代码实现2

```
const int maxn = 2000000 + 5;
int lens[maxn];
string expand(string s) {
    string ss = "$";
    for (int i = 0; i < s.size(); ++i) {
        ss += '#';
        ss += s[i];
    }
    ss += "#^";
    return ss;
}
```

```
int manachar(string s) {
    int mx = 0;
    int pos = 0;
    int ans = -1;
    for (int ptr = 1; ptr < s.size() - 1; ++ptr) {
        if (ptr < mx)
            lens[ptr] = min (lens[2 * pos - ptr], mx - ptr);
        else
            lens[ptr] = 1;
        while (s[ptr - lens[ptr]] == s[ptr + lens[ptr]])
            lens[ptr]++;
        if (mx < ptr + lens[ptr]) {
            mx = ptr + lens[ptr];
            pos = ptr;
        }
        ans = max (ans, lens[ptr] - 1);
    }
    return ans;
}
```

#3465. 「国家集训队」拉拉队排练

n 个人从左到右排成一行，每个人手中都举了一个写有26个小写字母中的某一个的牌子。如果连续的一段人，有奇数个，并且他们手中的牌子所写的字母，从左到右和从右到左读起来一样，那么这一段人就被称作和谐小群体。

现在想找出所有和谐小群体，并且按照人的个数降序排序之后，前 K 个和谐小群体的人个数的乘积是多少。（若 $< k$ 个 输出-1）

答案%19930726

$N=1e6$ $K=1e9$

一句话提炼问题：在长度为 n 的字符串中，求前 k 长的回文串长度之积。

#3465. 「国家集训队」拉拉队排练

这是一个板子题

Manacher 求每个位置的回文半径（注意batab 的情况 需要统计t ata 和 batab）

只求奇数长度，那么就直接求即可，不需要额外添加无关字符#

再利用桶统计一下即可。

再次温馨提醒，注意只统计奇数长度。

因为回文字符串包含关系，其统计过程类似前缀和。

#3465. 「国家集训队」拉拉队排练

N=1e6 K=1e9

这道题可能会在哪些细节上卡你？

- 1 快速幂
- 2 不开 long long 见祖宗
- 3 因为你没插# 所以之前的代码 `len[i]-1` 需要进一步修改

具体问题具体分析，不能乱套模版

#3466. 「国家集训队」最长双回文串

输入长度为 n 的串 S ，求 S 的最长双回文子串 T ，即可将 T 分为两部分 X, Y ， $(|X|, |Y| \geq 1)$ 且 X 和 Y 都是回文串。

$2 \leq \text{abs}(S) \leq 1e5$

Input: baacaabbaabab

out: 12

#3466. 「国家集训队」最长双回文串

求2个回文串拼一起

求len时再额外多维护两个信息

$L[i]$ 表示以 i 开头的最长回文串的长度。

$R[i]$ 表示以 i 结尾的最长回文串的长度。

$O(n)$ 再搜一遍即可。

和刚学LIS时的山峰题目类似。

练习时间

3478	「模板」Manacher 算法
2657	[POJ3974] Palindrome
3465	「国家集训队」拉拉队排练
3466	「国家集训队」最长双回文串
6724	回文匹配
6723	[THUPC2018]绿绿和串串
2460	[POI2010] Antisymmetry
6725	[JSOI2016] 扭动的回文串

建议复习一下hash