# Longest Palindromic Substring

In a string “abaaaab”, we get the LPS with “baaaab” where the substring is symmetric at center of “a”. The issue is that we should find the LPS length.

The initial idea to solve this issue is that loop each alphabet, for the specific alphabet, we compare left hand and right hand to calculate the LPS length. However, the computation complexity is so high. And we need to distinguish odd LPS and even LPS.

We import Manacher algorithm to solve this issue.

1) To unify odd and even LPS, we add special character in the string.

abaaaab => #a#b#a#a#a#a#b#. Then odd LPS “aba” becomes “#a#b#a#” => odd LPS.

Even LPS “baaaab” becomes “#b#a#a#a#a#b#” => odd LPS.

2) To reduce the computation complexity, a logic is applied:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| string | # | a | # | b | # | a | # | a | # | a | # | a | # | b | # |
| Len | 1 | 2 | 1 | 4 | 1 | 2 | 3 | 4 | 7 | 4 | 3 | 2 | 1 | 2 | 1 |

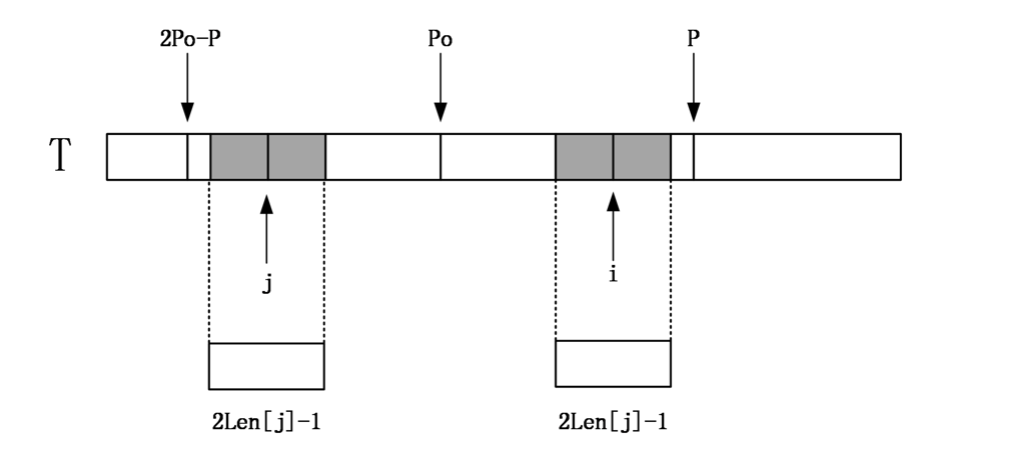
Define Len[i] is the left half LPS length including string[i] at the center of string[i]. Then LPS length of original string without extension is Len – 1.

### **Len数组的计算**

首先从左往右依次计算Len[i]，当计算Len[i]时，Len[j](0<=j<i)已经计算完毕。设P为之前计算中最长回文子串的右端点的最大值，并且设取得这个最大值的位置为po，分两种情况：

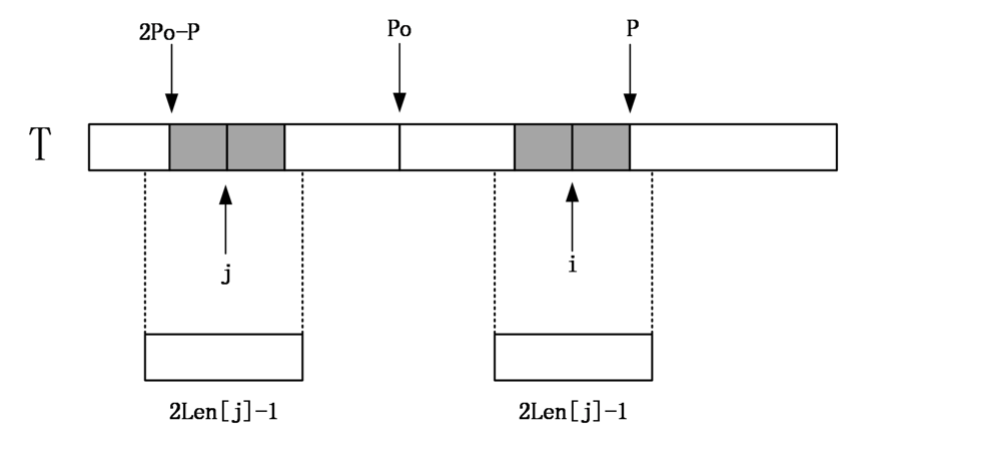
第一种情况：i<=P

那么找到i相对于po的对称位置，设为j，那么如果Len[j]<P-i，如下图：



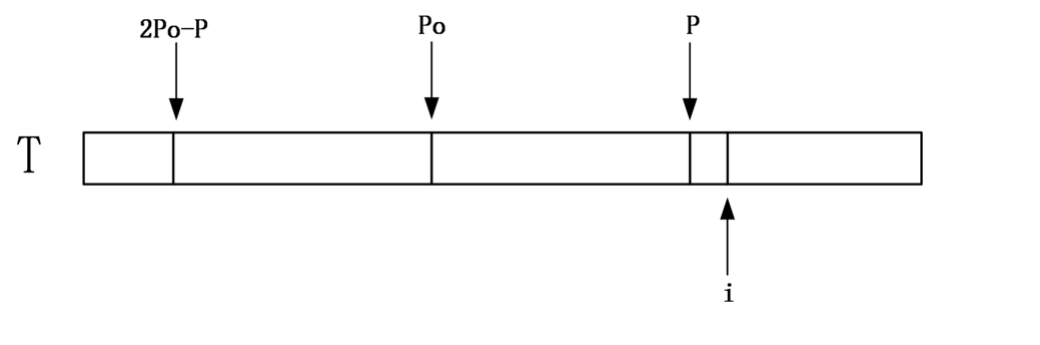
那么说明以j为中心的回文串一定在以po为中心的回文串的内部，且j和i关于位置po对称，由回文串的定义可知，一个回文串反过来还是一个回文串，所以以i为中心的回文串的长度至少和以j为中心的回文串一样，即Len[i]>=Len[j]。因为Len[j]<P-i,所以说i+Len[j]<P。由对称性可知Len[i]=Len[j]。

如果Len[j]>=P-i,由对称性，说明以i为中心的回文串可能会延伸到P之外，而大于P的部分我们还没有进行匹配，所以要从P+1位置开始一个一个进行匹配，直到发生失配，从而更新P和对应的po以及Len[i]。



第二种情况: i>P

如果i比P还要大，说明对于中点为i的回文串还一点都没有匹配，这个时候，就只能老老实实地一个一个匹配了，匹配完成后要更新P的位置和对应的po以及Len[i]。



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| string | # | a | # | b | # | a | # | a | # | a | # | a | # | b | # |
| Len | 1 | 2 | 1 | 4 | 1 | 2 | 3 | 4 | 7 | 4 | 3 | 2 | 1 | 2 | 1 |

len = strlen(string)

P0 = 0;

P = 0;

Len[0] = 1;

memset((void \*)Len, 1, 100 \* sizeof(int));

**for** (i=1;i<len;i++){

**if** (P<=i){

left = i;

right = i;

**while**(string[--left] == string[++right]){

Len[i]++;

}

P0 = i;

P = P0 + Len[i];

}**else**{

Temp\_Len = Len[2\*P0 - i];

**if** (Temp\_Len >= (P-i)){

Len[i] = P-i;

left = 2\*i-P;

right = P;

**while**(string[left--] == string[right++]){

Len[i]++;

}

}**else**{

Len[i] = Temp\_Len;

}

}

}