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

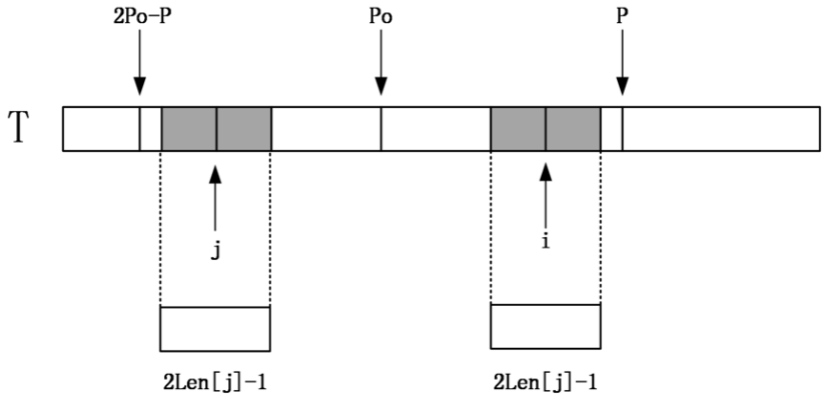
Define Len[i] is the left half LPS length including string[i] at the center of string[i]. Then LPS length of original string without extension is Len – 1.

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：

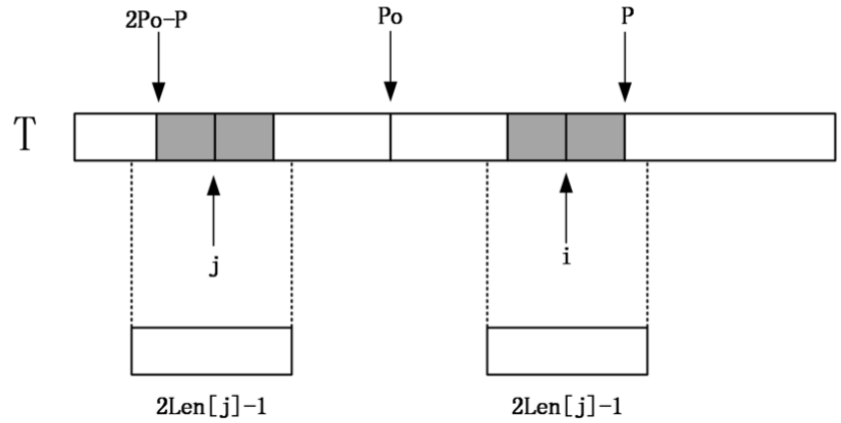
第一种情况：i<=P

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：



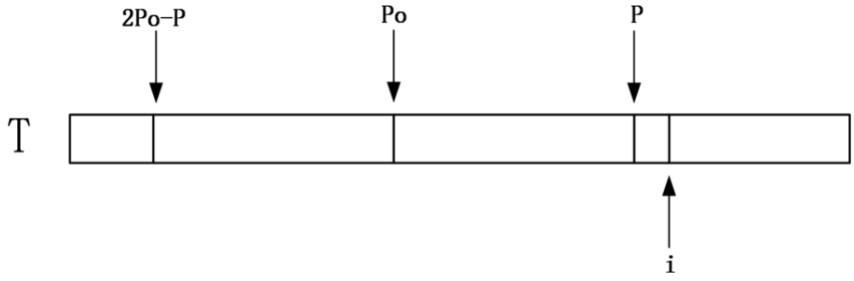
那么说明以 j 为中心的回文串一定在以 po 为中心的回文串的内部，且 j 和 i 关于位置 po 对称，由回文串的定义可知，一个回文串反过来还是一个回文串，所以以 i 为中心的回文串的长度至少和以 j 为中心的回文串一样，即 Len[i]>=Len[j]。因为 Len[j]<P-i,所以说 i+Len[j]<P。由对称性可知 Len[i]=Len[j]。

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;
                }
        }
}
```