Manacher算法

求最长回文子串

重庆南开信竞课程

1.问题背景

回文串与回文子串

回文串：如果一个字符串正着读和反着读是一样的，那它就是回文串。下面是一些回文串的实例：

A

ABA

ABBA

回文子串：如果一个字符串S的子串S’是回文串，我们称S’为S的回文子串。
其中长度最长的一个回文子串，称为S的最长回文子串。下面是一些最长回文子串的例子：

1. s=“abcd”，s的最长回文子串长度为 1；
2. s=“ababa”，s的最长回文子串长度为 5，就是s本身；
3. s=“abccbc”，s的最长回文子串长度为 4，即bccb。

求最长回文子串的传统思路是，遍历每一个字符，以该字符为中心向两边暴力查找。其时间复杂度为$O(n^2)$，效率很低。

1975年，一个叫Manacher的人发明了一个算法，Manacher算法（俗称：马拉车算法），该算法可以把算法效率提升到O(n)。

2. Manacher算法原理

奇回文和偶回文

由于回文分为偶回文（比如 abba）和奇回文（比如 aba），而在处理奇偶问题上会比较繁琐，所以这里我们使用一个技巧，具体做法是：在字符串首尾，及各字符间各插入一个字符（插入的这个字符是原串中没有的）。

```
aba  ——>  #a#b#a#
abba ——>  #a#b#b#a#
```

插入的是同样的符号，且符号不存在于原串，因此子串的回文性不受影响，原来是回文的串，插完之后还是回文的，原来不是回文的，依然不会是回文。

调整后，串的长度都转换成了奇数。

最长回文半径

我们把一个回文串中最左或最右位置的字符与其对称轴的距离称为回文半径。Manacher定义了一个**回文半径数组R**，用R[i]表示以第i个字符为对称轴的回文串的回文半径。

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
S	@	#	a	#	b	#	b	#	a	#	h	#	o	#	p	#	x	#	p	#
R[i]	边界	1	2	1	2	5	2	1	2	1	2	1	2	1	2	1	4	1	2	1

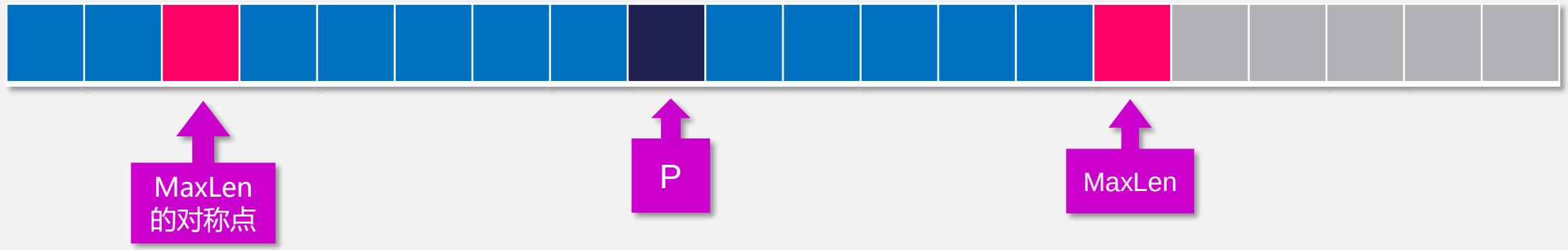
可以看出，R[i] - 1正好是原字符串中以i为中心的最长回文串的长度。
原串的最长回文串长度为max{ R[i]-1 }

于是问题变成了，**怎样高效地求的R数组。**基本思路是利用**回文串的对称性，扩展回文串**。

最长回文半径

我们从左往右依次讨论以每个字符为中心的回文半径的长度。

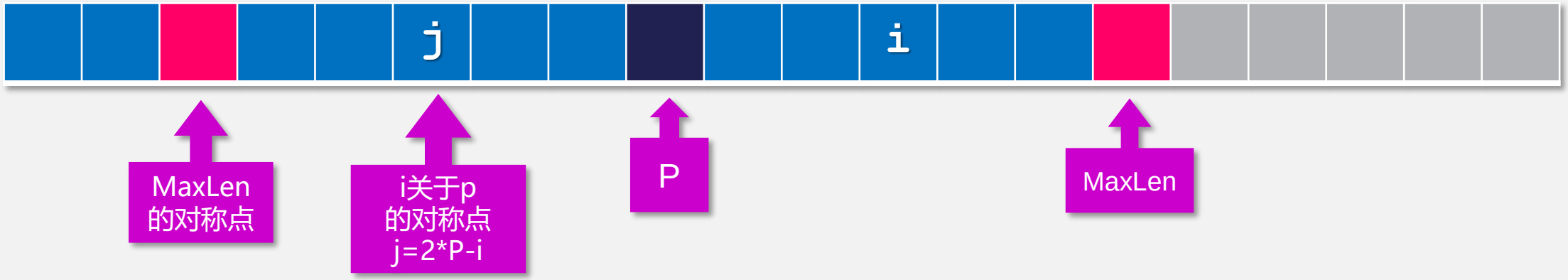
我们设MaxLen表示，当前已讨论过的回文子串中，最右边能到达的位置，同时用P记录下对应回文子串的中心所在位置。



我们从左往右依次求每个字符的R[ ]，假设当前访问到的位置为i，即要求R[i]，在对应上图，i必然是在P右边的。但我们更关注的是，i是在MaxLen的左边还是右边。我们需要分情况来讨论。

最长回文半径

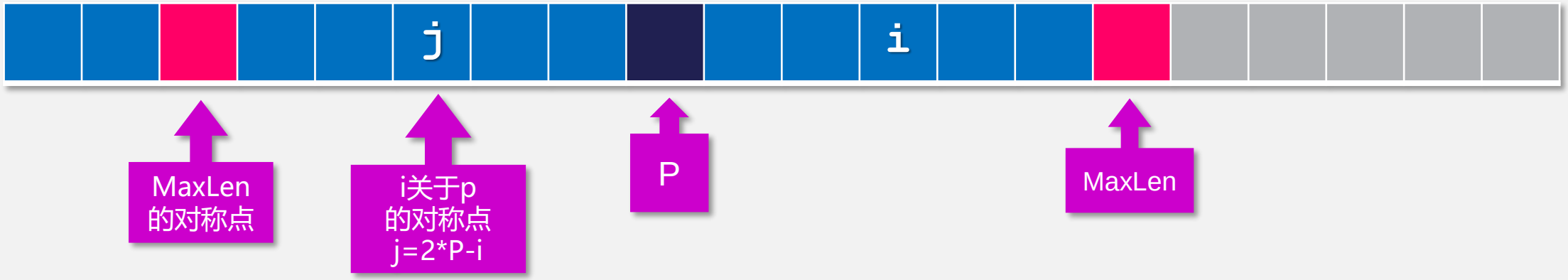
情况1：i在MaxLen的左边



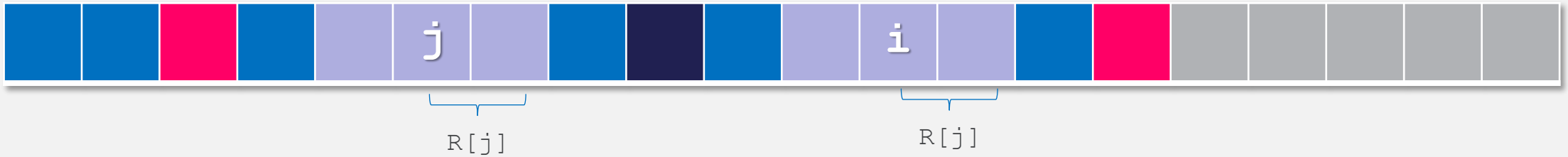
图中两个红色块之间（包括红色块）的串是回文；并且以i为对称轴的回文串，是与红色块间的回文串有所重叠的。
我们找到i关于P的对称位置j，这个j对应的R[j]我们是已经算过的。
根据回文串的对称性，以i为对称轴的回文串和以j为对称轴的回文串，有一部分是相同的。
这里又有两种细分的情况。

最长回文半径

情况1：i在MaxLen的左边



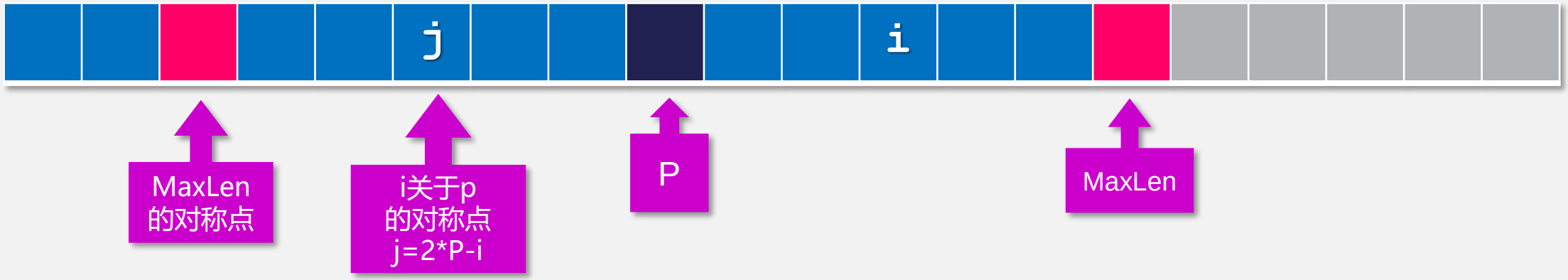
图中两个红色块之间（包括红色块）的串是回文；并且以i为对称轴的回文串，是与红色块间的回文串有所重叠的。
我们找到i关于P的对称位置j，这个j对应的R[j]我们是已经算过的。
根据回文串的对称性，以i为对称轴的回文串和以j为对称轴的回文串，有一部分是相同的。
这里又有两种细分的情况。**情况一：以j为对称轴的回文串比较短**



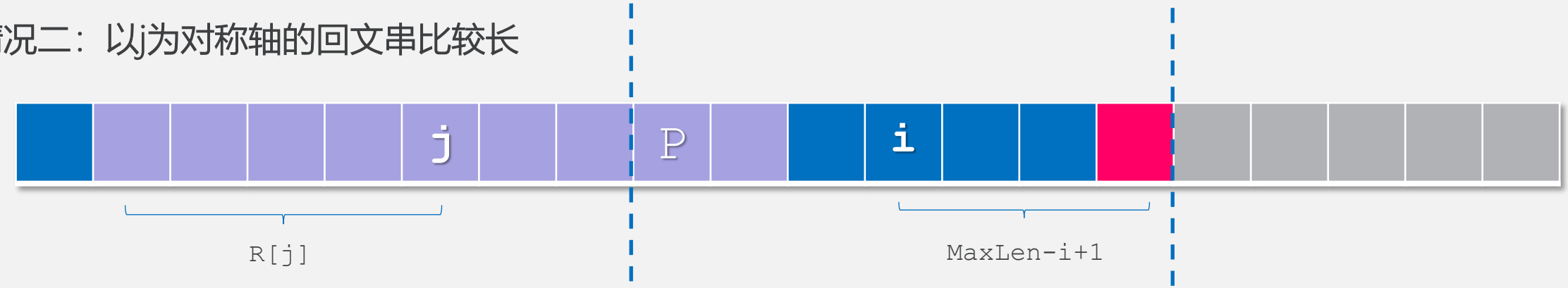
我们知道R[i]至少不会小于R[j]，于是可以令R[i]=R[j]。但是以i为对称轴的回文串可能实际上更长，因此我们试着以i为对称轴，继续往左右两边扩展，直到左右两边字符不同，或者到达边界。

最长回文半径

情况1：i在MaxLen的左边



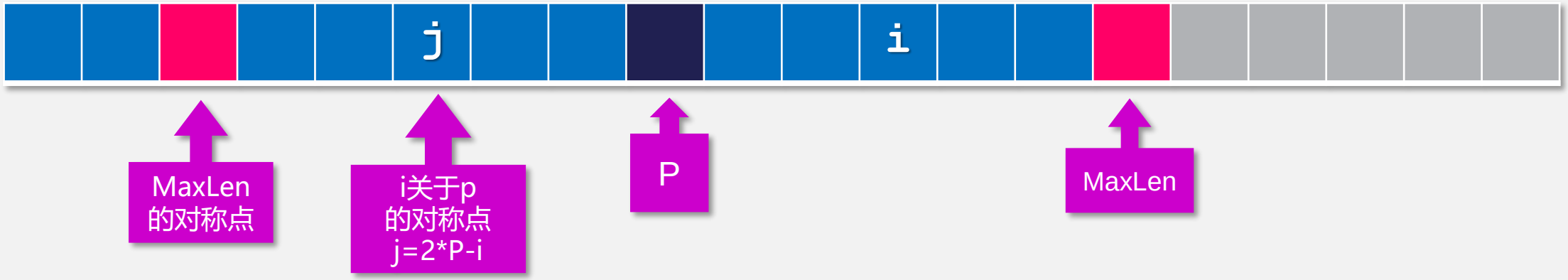
情况二：以j为对称轴的回文串比较长



此时，我们只能确定，两条蓝线之间的部分（即不超过MaxLen的部分）是回文，于是从这个长度(MaxLen-i+1)开始，尝试以i为中心向左右两边扩展，直到左右两边字符不同，或者到达边界。

最长回文半径

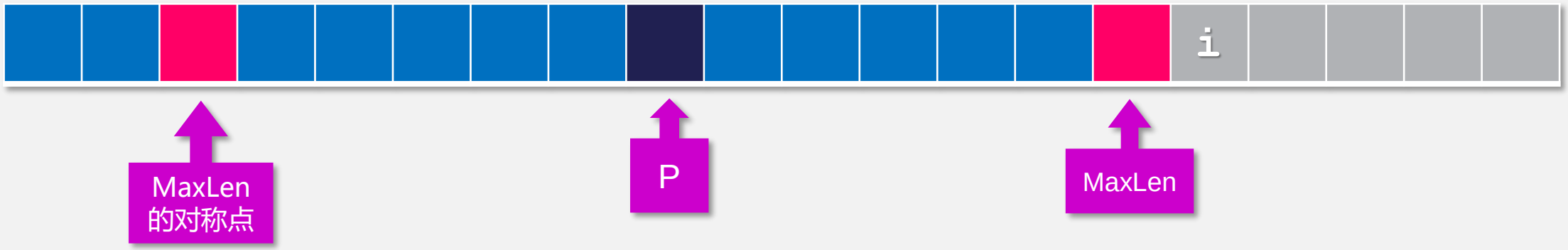
情况1：i在MaxLen的左边




```
step 1:   令R[i]=min(R[2*P-i], MaxLen-i+1)
step 2:   以i为中心扩展回文串，直到左右两边字符不同，或者到达边界。
step 3:   更新MaxLen和P
```

最长回文半径

情况2：i在MaxLen的右边



这种情况，说明以i为对称轴的回文串还没有任何一个部分被访问过，要计算R[i]就只能从i的左右两边开始尝试扩展了，当左右两边字符不同，或者到达字符串边界时停止。

```
step 1:   R[i]=1;
step 2:   while(s[i-R[i]]==s[i+R[i]])R[i]++;
step 3:   更新MaxLen和P
```

3. Manacher代码实现

参考代码

```
int Manacher()
{
    int len = Init();  // 在原串中插入特殊符号，并得到新字符串的长度len
    int Ans= -1;       // 最长回文长度
    int P=0;
    int MaxLen = 0;
    for (int i = 1; i < len; i++)
    {
        if (i < MaxLen)R[i] = min(R[2 * P - i], MaxLen - i+1);
        else R[i] = 1;
        while (S[i - R[i]] == S[i + R[i]])R[i]++;  //不用判断边界，在Init()中对边界做了处理
        if (MaxLen < i + R[i]-1)
        {
            P = i;
            MaxLen = i + R[i]-1;
        }
        Ans = max(Ans, R[i] - 1);
    }
    return Ans;
}
```

参考代码

```
int Init()
{
    int len = strlen(s);
    S[0] = '@';
    S[1] = '#';
    int j = 2;

    for (int i = 0; i < len; i++)
    {
        S[j++] = s[i];
        S[j++] = '#';
    }

    S[j] = '\0';  // 末尾添字符串结束符

    return j;     // 返回修改后的字符串的长度
}
```

4. Manacher时间复杂度

复杂度

时间复杂度：尽管代码里面有两层循环，但外层for循环执行次数为n次，内层while循环执行次数不会超过n次。

通过均摊时间分析(amortized analysis)我们可以得出，Manacher的时间复杂度是线性的。由于内层的循环只对尚未匹配的部分进行，因此对于每一个字符而言，只会进行一次，因此时间复杂度是**O(n)**。

课后习题：NKOJ 2517, 4347, 2663, 4716
BZOJ 2342, 2565, 3325

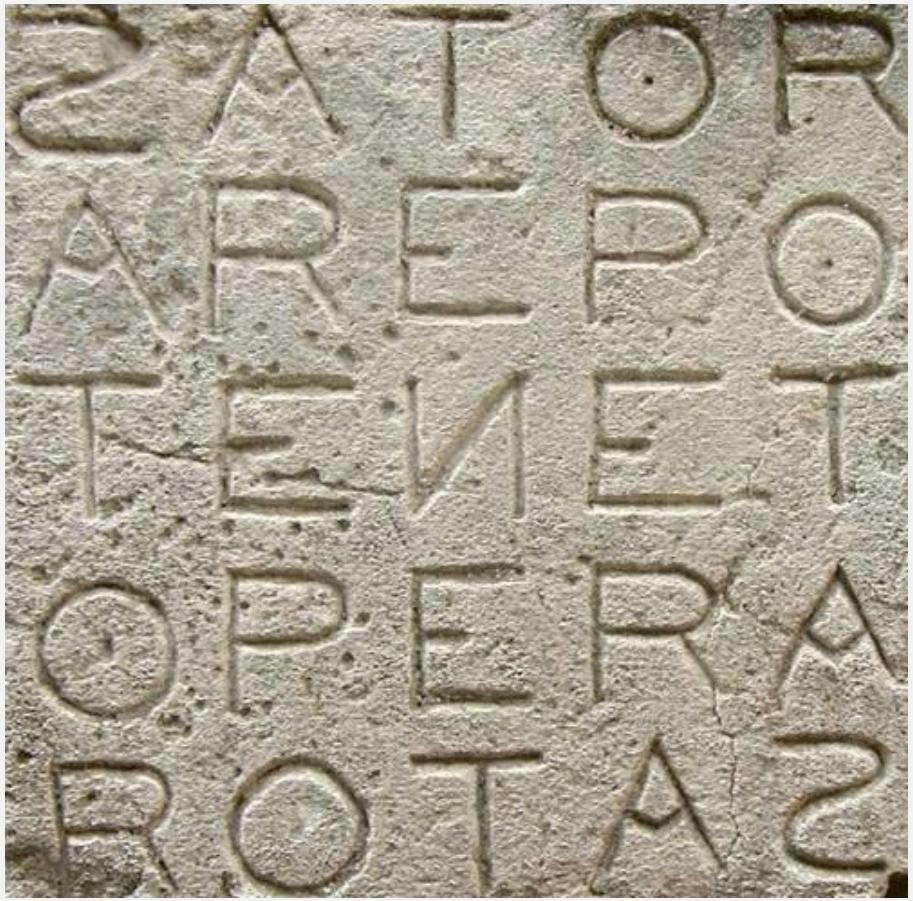
5. 回文串的一些趣味知识

观察下列古代遗迹的特点

赫库兰尼姆的古城遗迹中，有一个好玩的拉丁语回文串：sator arepo tenet opera rotas

“一个叫Arepo的**农夫**在播种，他用力扶着车轮”

Sator Square



它的第i行与第i列是相同的，该字符矩阵称为”Sator Square”

Sator Square

R O M A
O L I M
M I L O
A M O R

来自庞贝古城：

ROMA 罗马
OLIM 曾经
MILO 米罗(人名)
AMOR 爱

最长的回文英文单词

”tattarrattat”

出自爱尔兰作家詹姆斯·乔伊斯的小说《尤利西斯》，意思是敲门。

回文小说

《Satire: Veritas》

《Dr Awkward & Olson in Oslo》

A man, a plan, a caret, a ban, a myriad, a sum, a lac, a liar, a hoop, a pint, a catalpa, a gas, an oil, a bird, a yell, a vat, a caw, a pax, a wag, a tax, a nay, a ram, a cap, a yam, a gay, a tsar, a wall, a car, a luger, a ward, a bin, a woman, a vassal, a wolf, a tuna, a nit, a pall, a fret, a watt, a bay, a daub, a tan, a cab, a datum, a gall, a hat, h fag, a zap, a say, a jaw, a lay, a wet, a gallop, a tug, a trot, a trap, a tram, a torr, a caper, a top, a tonk, a toll, a ball, f fair, a sax, a minim, a tenor, a bass, a passer, a capital, a rut, an amen, a ted, a cabal, a tang, a sun, an ass, a maw, a sag, a jam, a dam, a sub, a salt, an axon, a sail, an ad, a wadi. a radian, a room, a rood, a rip, a tad, a pariah, a revel, a reel, a reed, a pool, a plug, a pin, a peek, a parabola, a dog, a pat, a cud, a nu, a fan, a pal, a rum, a nod, an eta, a lag, an eel, a batik, a mug, a mot, a nap, a maxim, a mood, a leek, a grub, a gob, a gel, a drab, a citadel, a total, a cedar, a tap, a gag, a rat, a manor, a bar, a gal, a cola, a pap, a yaw, a tab, a raj, a gab, a nag, a panan, a bag, a jar, a bat, a way, a papa, a local, a gar, a baron, a mat, a rag, a gap, a tar, a decal, a tot, a led, a tic, a bard, a leg, a bog, a burg, a keel, a doom, a mix, a map, an atom, a gum, a kit, a baleen, a gala, a ten, a don, a mural, a pan, a faun, a ducat, a pagoda, a lob, a rap, a keep, a nikp, a gulp, a loop, a deer, a leer, a lever, a hair, a pad, a tapir, a door, a moor, an aid, a raid, a wad, an alias, an ox, an atlas, a bus, a madam ,a jag, a saw, a mass ,an anus, a gnat, a lab, a cadet, an em, a natural, a tip, a caress, a pass, a baronet, a minmax, a sari, a fall, a ballot, a knot, a pot, a rep, a carrot, a mart, a part, atort, agut, a poll, a gateway, a law, a jay, a sap, a zag, a fat, a hall, a gamut, a dab, a can, a tabu, a day, a batt, a waterfall, a patina, a nut, a flow, a lass, a van , a mow, a nib, a draw, a regular, a call, a war, a stay, a gam, a yap, a cam , a ray, an ax, a tag, a wax, a paw, a cat, a valley, a drib, a lion, a saga, a plat, a catnip, a pooh, a rail, a calamus, a dairyman, a bater, a canal—Panama.

汉语回文

上海自来水来自海上
山东落花生花落东山
北京安定门定安京北
黄山落叶松叶落山黄
香山碧云寺云碧山香

人过大佛寺寺佛大过人
客上天然居居然天上客
僧游云隐寺
寺隐云游僧

——纪晓岚

奋斗吧少年

巨大的成功需要付出巨大的代价

no sacrifice, no success