寻找最大回文串。

填充数组和中心扩展算法。

```
#include <iostream>
#include <algorithm>
using namespace std; //问题背景，求出子串中的最大回文串。
int ans;
int main()
{
    string ss = "aaaaaa";
    //偶数串，奇数串分类。
    string s;
    s.resize(2 * ss.length() + 1);
    for (int i = 0; i < ss.length(); i++)
    {
        s[i * 2] = '#';
        s[i * 2 + 1] = ss[i]; //先填充。
    }
    s[2 * ss.length()] = '#';
    cout << s << '\n';
    for (int i = 1; i < s.length(); i++)
    {
        int l = i - 1, r = i + 1;
        bool flag = (s[i] == '#');
        while (true)
        {

            if (!(l >= 0 && r < s.length()) || s[l] != s[r])
            { //分类讨论中心位置字符的情况。这个会影响到
                ans = max(ans, r - i - 1);
                break;
            }
            r++, l--;
        }
    }
    cout << ans << '\n';
}
```

manacher

- ## 算法作用
  - 计算出每个位置为中心的最大回文串长度，复杂度仅为$o(n)$。
- ## 算法的设计思想
  - 关注到对称性。由已经计算的回文串中获取有效的信息。
- ## 算法描述；

- 涉及存储工具 $d(n)$),存储当前位置为中心，回文串的最大长度。 l，r， mid。记录r最远的对称信息。
- 判断查询位在范围之内
  - 若不在范围之内，直接进行中心扩展。
  - 若在范围之内，寻找对称位。利用对称位的可用信息。详情分两种情况。
    - **TYPE ONE　可处理可利用的信息对称位的回文串范围，大于等于对称边界，初步确定d[i],然后继续向下暴力拓展。**
    - **TYPE TWO　可确定对称范围小于边界，此时，直接确定d[i]即可。**
- 每计算完一个查询位，尝试更新右边界最远的对称区间。

不填充式manacher实现。

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
string s;
vector<int> d1;
vector<int> d2;
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    cin >> s;
    int n = s.size();
    d1.resize(n);
    for (int i = 0, l = 0, r = -1, k; i < n; i++)
    {
        k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
        while (i - k >= 0 && i + k < n && s[i - k] == s[i + k])
            k++;
        d1[i] = k--;
        if (i + k > r)
            l = i - k, r = i + k;
    }
    d2.resize(n); //偶数这里的额意义是是是什么？
    for (int i = 0, l = 0, r = -1, k; i < n; i++)
    {
        k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
        while (i - k - 1 >= 0 && i + k < n && s[i - k - 1] == s[i + k])
            k++;
        d2[i] = k--;
        if (i + k > r)
        {
            l = i - k - 1;
            r = i + k;
        }
    }
```

```
    int ans = 0;
    for (int i = 0; i < n; i++)
    {
        ans = max(ans, d1[i] * 2 - 1);
        ans = max(ans, d2[i] * 2);
    }
    cout << ans << '\n';
}
```

快速回顾tips;

- $d1(maxn)$**指的是奇数回文串半径。中心到边界长度。**
- $d2(maxn)$**指的是偶数回文串的径。上述代码定位中，凭借中心线后一位元素为参考点进行拓展。**

自己的sb式代码。(不简洁，if叠太多)

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <algorithm>
using namespace std; //问题背景，求出子串中的最大回文串。
int ans, n;
string s;
vector<int> d1; //处理偶数串
vector<int> d2; //处理奇数串
int main()
{
    cin >> s;
    n = s.length();    //长度
    int l = 0, r = -1; //由于初始条件下，不存在对称区间，防止干扰。于是设置为r=-1。
    d1.resize(n);
    d2.resize(n);
    for (int i = 0, t; i < n; i++) // t是当前探索的长度。
    {
        t = 1;
        if (i > r) //不在范围内这里做暴力的中心拓展。
        {
            while (i + t - 1 < n && i - t >= 0 && s[i - t] == s[i + t - 1]) //这里将进行暴力拓展。
                t++;
            d1[i] = --t;
        }
        else
        {
            int zoz = l + r - i + 1; //对称点。
            if (i + d1[zoz] < r)     //对称点边界在范围内。
            {
```

```
                d1[i] = d1[zoz];
                continue;
            }
            else //对称点边界>=范围内。等于的时候也算。
            {
                t = d1[zoz];
                while (i + t -1< n && i - t >= 0 && s[i - t] == s[i + t - 1]) //
这里将进行暴力拓展。
                    t++;
                d1[i] = --t;
            }
        }
        if (i + d1[i] - 1 > r && d1[i] != 0) //尝试更新对称区间。//边界依然不存在要特
判。
        {
            r = i + d1[i] - 1;
            l = i - d1[i];
        }
    }
    //奇数处理和偶数处理大体相似。差别仅仅在于，定位中点的不同。计算定位左右边界的式子有点不同。
    l = 0, r = -1;
    for (int i = 0, t; i < n; i++) // t是当前探索的长度。
    {
        t = 1;
        if (i > r) //不再范围内
        {
            while (i + t < n && i - t >= 0 && s[i - t] == s[i + t]) //这里将进行暴
力拓展。
                t++;
            d2[i] = --t;
        }
        else
        {
            int zoz = l + r - i;    //对称点。
            if (zoz - d2[zoz] > l) //对称点边界在范围内。
            {
                d2[i] = d2[zoz];
                continue;
            }
            else //对称点边界不在范围内。
            {
                t = d2[zoz];
                while (i + t < n && i - t >= 0 && s[i - t] == s[i + t]) //这里将进
行暴力拓展。
                    t++;
                d2[i] = --t;
            }
        }
        if (i + d2[i] > r) //尝试更新对称区间。
        {
            r = i + d2[i];
            l = i - d2[i];
        }
    }
    for (int i = 0; i < n; i++)
```

```
        ans = max(d1[i] * 2, max(ans, d2[i] * 2 + 1));
    cout << ans << '\n';
}
```

问题以及总结：

- **追求更简洁。**
  - **哪些步骤是多余的?**
  - **if是否可以改为条件选择语句?**
    - 上面的偶数串或者奇数串中，第一个分流上；看现在的解决的中心在不在区间上。这里可以直接用一个，条件选择语句来完成。
    - 第二个看边界有对称点的回文边界有没有超出，也可以直接使用条件选择语句完成。
  - **几个if中有什么相同的代码。把他们分出来到主干上。**
    - 第一份代码，不同的三种情况，只是影响了，下一步拓展得位置。无论是哪种情况，while中心拓展后都不会有影响。所以可以考虑直接把一大堆if删掉。
  - **if语句**
    - 代码写长。
    - 当判断太多时，队友自己看不懂，也容易写错。
    - 普通写法，就是暴力得实现。而没有精巧得思考。我唾弃这种方　式。
- **一个非常细节的地方。**
  - 偶数串，中新参考点的位置，要设置为中心线后后面的元素。否则会出现，非常容易对称点信息利用错误的问题。思考一下为什么，可以重利用，如果设置为中心线前，画一画即可。
  - 测试例子：　　$baaccaabaccaab$

下面是加入了填充数组的代码。$final-mostclearest$

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <algorithm>
using namespace std; //问题背景，求出子串中的最大回文串。
string s1, s;
int ans;
vector<int> d; //处理偶数串
int main()
{
    cin >> s1;
    int n = s1.size();
    s.resize(n * 2 + 1);
    for (int i = 0; i < n; i++)
    {
        s[i * 2] = '#';
        s[i * 2 + 1] = s1[i];
    } //于是已经构建好了。
    s[n * 2] = '#';
    d.resize(n * 2 + 1); //
```

```
    n = n * 2 + 1;
    for (int i = 1, l = -1, r = -1, k; i < n; i++)
    {

        k = i > r ? 1 : min(d[l + r - i], r - i);
        while (i - k >= 0 && i + k < n && s[i + k] == s[i - k])
            k++;
        d[i] = --k;
        if(i+k>r)
            r = i + k, l = i - k;
        ans = max(d[i], ans);
    }
    cout << ans;
}
```

快速回顾tips:

- $d(i)$ ： 以i为中心的半径，不包括中心这个元素。
- r, l。先设置为-1.
- k，代表当下对比的半径，用于辅助判断r。
- 对称点的计算过程如下：
  $$mid = \left[\frac{l+r}{2}\right] + 1 = \frac{l+r}{2} - 0.5 + 1 = \frac{l+r}{2} + 0.5$$
  $$dui - chen - dian = mid - (i - mid) = 2 \times mid - i = l + r - i$$

2022 8 11 12：25在信宜