# 转载：http://blog.csdn.net/cn\_wk/article/details/52541367

应用：假设一个乱序数组，需要查找一个元素是否在该数组中，这时需要用到顺序查找，也就是遍历数组。

一般情况下我们会写下如下代码：

**[cpp]** [view plain](http://blog.csdn.net/qiuchenl/article/details/9954837) [copy](http://blog.csdn.net/qiuchenl/article/details/9954837)

1. **int** Sequential\_Search(**int** \*a,**int** n,**int** key)
2. {
3. //数组从1开始
4. **int** i;
5. **for**(**int** i=1;i<=n;i++)
6. {
7. **if**(a[i]==key)
8. **return** i;
9. }
10. **return** 0;//查找失败
11. }

有的数据结构书上，会运用哨兵元素，改成这样的代码：

**[cpp]** [view plain](http://blog.csdn.net/qiuchenl/article/details/9954837) [copy](http://blog.csdn.net/qiuchenl/article/details/9954837)

1. **int** Sequential\_Search2(**int** \*a **int** n,**int** key)
2. {
3. **int** i=0;
4. a[0]=key;//哨兵
5. i=n;
6. **while**(a[i]!=key)
7. {
8. i--;
9. }
10. **return** i;//返回0就是查找失败
11. }

仔细看来没有什么差别，但是来看下我测试的运行时间，数组有10亿个元素。

方案1：3.494s   3.202s   3.216s   3.237s

方案2：2.332s   2.307s   2.24s   2.194s

为什么基本一样的代码，方案2比方案1性能提升了30%~40%左右？？？

循环中，方案1有3条指令而方案2有两条指令，少了i<n这个比较操作，所以方案2性能得到了提升，这也是哨兵元素的妙用

以上思想和代码来自于《大话数据结构》中的296页，测试实验是我做的。

大话数据结构的原文“这种查找方法在查找方向的尽头设置哨兵元素，免去了查找过程中每次比较后都要判断查找位置是否越界的小技巧，看似与原先差别不大，但是总数据较多时，效率提高很明显，是非常好的编程技巧。当然，“哨兵”也不一定在数组开始，也可以再末尾”

我的测试程序：

**[cpp]** [view plain](http://blog.csdn.net/qiuchenl/article/details/9954837) [copy](http://blog.csdn.net/qiuchenl/article/details/9954837)

1. **void** main()
2. {
3. **int** num=1000000000;
4. **char** \*p=**new** **char**[num];
5. p[0]=2;
6. **char** key=p[0];
7. **clock\_t** start, finish;
8. start=clock();
9. **if**(**true**)
10. {
11. **for**(**int** i=1;i<num;i++)
12. {
13. **if**(p[i]==key)
14. **break**;
15. }
16. }
17. **else**
18. {
19. **int** i=num-1;
20. **while**(p[i]!=key)
21. {
22. i--;
23. }
24. }
25. finish=clock();
26. **double** Total\_time = (**double**)(finish-start) / CLOCKS\_PER\_SEC;
27. cout<<Total\_time<<endl;
28. }