704. 二分查找

前言

```
1  第一次写LeetCode 题解，纪念一下，会越来越好的
2  刚学算法不久，题解难免有错误，仅供参考
```

题目

```
1  给定一个 n 个元素有序的（升序）整型数组 nums 和一个目标值 target  ，写一个函数搜索 nums 中的 target，如果目标值存在返回下标，否则返回 -1。
```

解题思路

```
1  本题比较简单，是二分查找算法的基本应用：
2  二分查找又名折半查找，算法如其名字一样，当我们在一个有序的数组中查找的时候，我们可以首先判断中点关键值，如果中点值比需要查找的值小，我们就可以确定我们需要查找的值，在右边的区域，反之在左边区域，通过这样的思路，不断迭代，就可以确定我们需要查找的值的下标
```

C++ 实现二分查找

```cpp
class Solution {
public:
    int search(vector<int>& nums, int target) {
        // 确定左右哨兵 [mi,hi)
        int lo = 0,hi = nums.size();

        while (lo < hi){
            int mi = (lo + hi) >> 1; // >> 1  等价于 /2 相对来说速度稍快

```

```
            if (target < nums[mi]){
                hi = mi;
            }
            else if (nums[mi] < target){
                lo = mi + 1;
            }
            else {
                return mi;
            }
        }

        // 只要命中就会在前面退出，如果退出循环标识不存在 返回 -1
        return -1;
    }
};
```

优化 Fibonacci 查找

为什么需要优化

```
我们发现二分查找转向左分支只需要一次比较 即 target < nums[mi]
转向右分支却需要两次，我们能否减少转向右分支的次数来优化时间复杂度呢?
换句话说能否让这颗查找树向左倾斜?

我们采用Fibnacci查找
也就是让左侧区域的长度接近右侧区域的长度的两倍，来补偿这一比较差异
```

创造Fibnacci类

```
class Fib{
// _prev 记录当前项（_hot）的前一项，_hot为当前项
    int _prev,_hot;
public:
    // 初始化
    Fib(int target){
```

```
7           // 初始化Fibnacci 函数前两项
8           _prev = 0;
9           _hot = 1;
10          // 当当前项小于目标值，不断向前迭代
11          while(get() < target){
12              next();
13          }
14      }
15      // 返回当前项
16      int get(){
17          return _hot;
18      }
19      // 迭代到下一项返回
20      int next(){
21          // 生成下一项
22          int newHot = _hot + _prev;
23
24          _prev = _hot;
25          _hot = newHot;
26
27          return get();
28      }
29      // 往前回退一项
30      int prev(){
31          // 退回前面一项
32          int prev_prev = _hot - _prev;
33
34          _hot = _prev;
35          _prev = prev_prev;
36
37          return get();
38      }
39  };
```

Fibnacci 查找算法

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int lo = 0,hi = nums.size();

        Fib fib(hi - lo); // 初始化Fibnacci数列到数组长度

        while (lo < hi){
            // 不断将 fib数组缩小到界限内
            while (hi - lo < fib.get()){
                fib.prev();
            }

            // 获得下一个判断点
            int mi = lo + fib.get() - 1;

            if (nums[mi] < target){
                lo = mi + 1;
            }
            else if (target < nums[mi]){
                hi = mi;
            }
            else {
                return mi;
            }
        }

        return -1;
    }
};
```

借用邓俊辉在数据结构课程内的结论，Fibnacci查找在常系数意义上的改进已经达到极限

转向代价平衡的二分查找

```
1  既然Fibnacci查找做了如此多的努力，就是为了让转向的代价更多的趋于平衡，那么我们能否直接平衡转向比较代价吗?
2  我们能否去掉判断 直接相等的条件也就是最后的else，使其在一次判断后直接进去左，右分区，从而使得整一个循环的出口唯一，也就是 循环
```

```
1  class Solution {
2  public:
3      int search(vector<int>& nums, int target) {
4          // 确定左右哨兵 [mi,hi)
5          int lo = 0,hi = nums.size();
6
7          while (lo < hi){
8              int mi = (lo + hi) >> 1;
9              if (target < nums[mi]){
10                 hi = mi;
11             }
12             else {
13                 lo = mi + 1;
14             }
15         }
16         // 最后一定会锁定到 一个不大于target数到最后一个然后 lo = mi + 1,所以我们只要退一步就是我们需要的值
17         // 因为需要判断 lo-1 有可能lo == 0，所以排除一下特殊情况
18         return lo-1 >= 0 && nums[lo-1] == target ? lo-1 : -1;
19     }
20 };
```

复杂度

```
我们不难发现减少了最后一次判断带来了一个优点和一个缺点
优点：  左右转向代价完全平衡
缺点：  即使等于也不能直接退出，仍然要进行判断，直到区间长度为1才可以退出

我们不做过于复杂的复杂度分析，直接邓俊辉老师在数据结构课程的结论
相对于之前的版本，最好的情况性能会下降，最坏的情况会更好，各种情况下的SL更加接近，整体性能会更加趋于稳定。
```