算法设计、正确性证明、复杂度分析、编码和测试
–以二分查找为例

乔海燕
中山大学数据科学与计算机学院

2017 年 10 月 18 日

摘　要

本文以二分查找为例，说明算法的设计、证明、分析、编码和测试。

§1　二分查找算法

假设$L = (a_1, a_2, \cdots, a_n)$是长度为$n$的从小到大排列的元素序列，要查找$x$是否在该序列中出现。二分查找的方法：

- 当$n = 0$时, $L = ()$为空序列，故x在L中不出现；
- 当$n = 1$时, $L = (a_1)$，如果$x = a_1$成立，则返回存在，否则，返回不存在。
- 当$n > 1$时，取L的中间元素a_m, 其中$m = \lfloor \frac{1+n}{2} \rfloor$，如果$x > a_m$，则继续到后半部分$L_2 = (a_{m+1}, a_{m+2}, \cdots, a_n)$ 查找，否则$x \le a_m$, 因此到前半部分$L_1 = (a_1, a_2, \cdots, a_m)$查找。

由此可以设计递归的二分查找算法1，BinarySearch。由于查找过程中需要说明查找范围，所有算法中需要表达查找范围$[bot, top]$的参数bot和top。

由于算法1的空间复杂度为$O(\log n)$，可以设计空间复杂度为$O(1)$的迭代算法2。

§2　二分查找算法的正确性证明

定理1. *算法$BinarySearch$总是终止的，而且如果x在$A[bot..top]$中出现，则算法返回一个$bot \le i \le top$使得$A[i] = x$，否则返回-1。*

证明1. 对区间$[bot, top]$的长度进行归纳。如果区间$[bot, top]$为空（$top < bot$）则x不出现，算法返回-1，定理成立。如果区间$[bot, top]$ 长度为1（$bot = top$），则根据算法，如果x在该区间出现，则必有$A[bot] = x$，算法返回bot，如果x在该区间不出现，则$A[bot] \ne x$，算法返回-1，因此，此时定理仍然成立。

假设当区间$[bot, top]$长度小于m时定理成立，现证当区间$[bot, top]$长度为m时定理仍然成立。根据算法，首先计算$mid = [(bot + top)/2]$，根据x与$A[mid]$的比较结果分两种情况。

算法 1 BinarySearch(A, x, bot, top)

输入: A是长度为$n > 0$的整数数组, bot和top是数组的两个下标，x是一个元素。

输出: 如果$A[bot..top]$中存在$A[i] = x$, 则返回下标i, 否则返回-1。

```
if bot > top then
  return -1
if bot = top then
  if A[bot] = x then
    return bot
  else
    return -1
if bot < top then
  mid ← (bot + top)/2
  if x > A[mid] then
    return BinarySearch(A, x, mid + 1, top)
  else
    return BinarySearch(A, x, bot, mid)
```

算法 2 BinarySearch_iterative(A, x)

输入: A是长度为$n > 0$的整数数组, s是一个元素。

输出: 如果$A[0..n-1]$存在$A[i] = x$, 则返回下标i, 否则返回-1。

```
bot ← 0
top ← n − 1
while bot < top do
  mid ← (bot + top)/2
  if x > A[mid] then
    bot ← mid + 1
  else
    top ← mid
if top < bot then
  return -1
if bot = top then
  if A[bot] = x then
    return bot
  else
    return -1
```

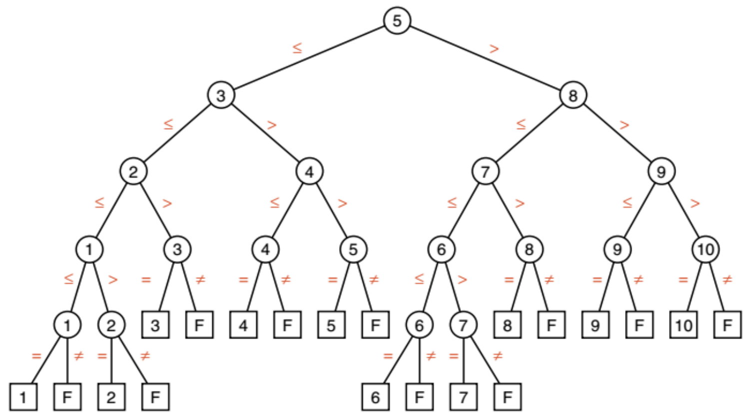

图 1: 二分查找BinarySearch的比较树，查找区间是[1, 10]。插图来自[2]。

1. $x > A[mid]$，结果是*BinarySearch*$(A, x, mid+1, top)$中。因为$bot < top$，因此$bot \le mid < top$，因此递归调用的区间$[mid+1, top]$ 长度小于区间$[bot, top]$的长度m，根据归纳假设，算法会终止。而且，x在$[bot, top]$出现当且仅当x在$[mid+1, top]$ 中出现，根据归纳假设，如果x在$[mid+1, top]$出现，则算法返回某个$bot \le mid+1 \le i \le top$使得$A[i] = x$，或者返回*-1*表示不存在。

2. 如果$x \le A[mid]$，则结果是*BinarySearch*(A, x, bot, mid)。同理，因为$bot < top$，$bot \le mid < top$，因此递归调用的区间$[bot, mid]$ 长度小于区间$[bot, top]$的长度m，根据归纳假设，算法会终止。而且，x在$[bot, top]$出现当且仅当x在$[bot, mid]$中出现，根据归纳假设，如果x在$[bot, mid]$出现，则算法返回某个$bot \le i \le mid \le top$使得$A[i] = x$，或者返回*-1*表示不存在。

因此，定理得证。

§3 二分查找的复杂度分析

二分查找的时间复杂度用$T(n)$表示，其中n是区间$[bot, top]$的长度，则有

$$T(n) = \begin{cases} 1 & \text{如果} n = 1 \\ T(n/2) + 1 & \text{否则} \end{cases}$$

解递推方程得到$T(n) = O(\log n)$。

空间复杂度为递归调用深度，故$S(n) = O(\log n)$。

另一方面，时间复杂度也可用比较树[2]分析。例如，$n = 10$时得到图1的比较树。

可以证明以下定理：

定理2. *假设二分查找BinarySearch在长度为n的区间上进行查找，则其比较树有如下特点：*

1. *比较树是一颗满二叉树，即每个内部结点有两个子结点；*

2. *每个叶结点表示一种查找结果，算法的每次运行是从根结点到某个叶结点路径上的关键字比较过程；*

3. 叶结点数为$2n$个，而且查找成功和查找失败情况各为n个；

4. 所有叶结点都在同一层或者相邻两层。确切地说，所有叶结点在$\lceil \log_2 2n \rceil$层或者$\lfloor \log_2 2n \rfloor$层。

证明略去。

定理说明，比较树的高度为$\lceil \log_2 2n \rceil$，因此，二分查找的最坏情况和平均情况时间复杂度均为$\lceil \log_2 2n \rceil$, 即$T(n) = O(\log n)$, 这对查找成功和查找失败均成立。

§4 二分查找的C++实现和测试

容易将以上递归伪代码转换为C++代码：

```
typedef int T;

int BinarySearch(vector<T> list, int bot, int top, T x){
  if (bot < top) {
     size_t mid = (bot + top)/2;
     if (list[mid] < x)
        return BinarySearch(list, mid+1, top, x);
     else
        return BinarySearch(list, bot, mid, x);
  }
  if (top < bot)
     return -1;
  else // bot == top
     if (list[bot] == x)
        return bot;
     else
        return -1;
}
```

根据二分查找的正确性命题，我们设计一个随机测试程序，方法为

- 随机生成一个长度为n的整数向量$list$，并将$list$排序；
- 对于$list$的每个元素x，令k=BinarySearch($list$, 1, n−1, x)，如果$list[k] \neq x$，则报告错误；
- 随机生成一些在$list$中不出现的x，如$x < list[0]$，或$x > list[n-1]$，或者对某个$0 \leq i < n-1$, $list[i] < x < list[i+1]$，令k=BinarySearch($list$, 1, $n-1$, x)，如果$k \neq -1$, 则报告错误。

```
int main(int argc, char *argv[]){
   if (argc < 2) {
      cout <<"usage: testBinarySearch n";
```

```
        exit(0);
    }
    int n = atoi(argv[1]);
    vector<int> v(n);
    int N = 100;
    for (int i=0; i<n; i++) {
        v[i] = rand()%N;
    }
    sort(v.begin(), v.end());
    cout << "the list: "; //输出有序向量
    for (size_t i=0;i<v.size();i++)
        cout <<v[i]<<" ";
    cout <<endl;
    //成功查找测试
    for (int j = 0; j<n; j++) {
         int k = binary_search(v, 0, n-1,v[j]);
         if (v[k] != v[j]){
              cout<<"Obs! "<<v[j]<<" not found!"<<endl;
              break;
         }
    }
    vector<int> u;
    u = missingFrom(v); //u是由不包含在v中元素构成的向量
    //失败查找测试
    for (int j = 0; j<n; j++) {
        int k = binary_search(v, 0, n-1,u[j]);
        if (k != -1){
            cout<<"Obs! "<<u[j]<<" is not there, but is found!"<<endl;
            break;
        }
    }
  return 0;
}
```

其中函数missingFrom(v)生成有序向量v缺失的部分元素：

```
vector<int> missingFrom(const vector<int> &v){
     vector<int> u;
     int x;
     x = v.front() - 1 - rand()% 100;//小于v.front()的元素
     u.push_back(x);
     for (size_t i=0;i<v.size()-1;i++){
         int x = v[i] + 1;
         while (x < v[i+1])
             u.push_back(x++); //介于v[i]和v[i+1]之间的元素
```

```
    }
    x = v.back() + 1 + rand()%100; //大于v.back()的元素
    u.push_back(x);
    return u;
}
```

鸣谢：感谢16级计算机专业同学们给予老师的激励，并指出文中的错漏问题！

参考文献

[1] 严蔚敏、吴伟民，数据结构，清华大学出版社，1997。

[2] Robert L. Kruse, Alexander J. Ryba. *Data Structures and Program Design in C++*, Higher Education Press, 2001.

[3] Michael T. Goodrich, Roberto Tamassia. *Algorithm Design and Applications*, Wiley, 2014.