13th July 2013

Binary search and its variation

1. Introduction

Writing correct programs is not an easy task, especially for some problems that require particularly careful code, like binary search. Knuth points out that while the first binary search was published in 1946, the first published binary search without bugs did not appear until 1962. There are a couple of binary search related problems in leetcode [http://leetcode.com/onlinejudge] . And several anonymous coders share awesome programs in the leetcode discussion forum [http://discuss.leetcode.com/] for these problems. Standing on the shoulder of these anonymous heroes and programming pearls [http://netlib.bell-labs.com/cm/cs/pearls/] , I'm trying to give a **14 lines** framework of writing correct programs for binary search problem and its variations and explain its correctness.

2. Framework

The following code snippet is the framework. I'll use two problems to explain it later.

```
int binarySearchFramework(int A[], int n, int target) {
    int start = start index of array - 1;
    int end = length of the A;
    while (end - start > 1) {
        int mid = (end - start) / 2 + start;
        if (A[mid] ? target) {
            end = mid;
        } else {
            start = mid;
        }
    }
    return ?;
}
```

(1) The first example : search insert position [http://leetcode.com/onlinejudge]

Given a sorted array with length *n*, saying *A* and a *target* value, return the index if the target is found. If not, return the index where it would be if it were inserted in order. You may assume no duplicates in the array. Here are few examples. [1,3,5,6], 5 → 2; [1,3,5,6], 2 → 1; [1,3,5,6], 7 → 4; [1,3,5,6], 0 → 0

Let's see how our framework can be used to solve this problem. We use *start* and *end* indices into the array that bracket the position of *target*. So for this problem, the invariant relation is *A[start] < target <= A[end]* [http://www.blogger.com/blogger.g?blogID=3437517718391875085] . If anyone has no idea of what invariant relation is, I highly recommend Column 4 of Programming pearls [http://www.it.iitb.ac.in/~deepak/deepak/placement/Programming_pearls.pdf] .

This invariant relation will guide us filling "?" in the framework.

a. Initialization

How to initialize *start* and *end* to keep the invariant relation? It's a problem. ☺. Someone may think *start* starts from 0 and she/he will realize that it's wrong after thinking in a couple of seconds since *target* can be less than *A[0].* The magic is that initializing *start* as -1(it must be -1, no other choices) and we assume A*[-1] < target*. Leave a question for the reader, why *start* can't be -2, -3, …? And for the *end*, I guess smart as you have already got it: *end = n*(it must be n, no other choices) and we assume *target <= A[n].*

b. Maintaining invariant relation inside the loop

Using ***A[mid] >= target*** can maintain the invariant relation. Take a close look at the framework:

```
if (A[mid] >= target) {
    end = mid;
} else {
    start = mid;
}
```

1) If A*[mid] >= target*, assign *end* as *mid*. So *A[mid] >= target* guarantees *A[end] >= target* which exactly matches our invariant relation.

2) If *A[mid] < target*, assign *start* as *mid*. So *A[mid] < target* guarantees *A[start] < target* which exactly matches our invariant relation.

c. Return

After quitting from the loop, our invariant relation *A[start] < target <= A[end]* still maintains since we don't modify it during the loop. If *target == A[end],* we should return *end* with no doubt. If *A[start] < target < A[end],* then the insert position is *end*.

d. Halting proof

At last, we have to prove that the loop will terminate(it will not become an infinite loop). I'll use inductive reasoning to explain why it will terminate at the end.

1) For the most basic case: the length of the array is 1(*n = 0* is undefined for this problem, so we pass it), if you simulating the code running, it will terminate.
2) For *n* is 2, after executing *mid = (end – start)/2 + start*, both *A[start, mid]* and *A[mid, end]* can be seen as a basic case(*n = 1*). Since we have already proved that the basic case *n = 1* will terminate. So *n =2* will also terminate.
3) *N = 3* can be seen as a basic case and a case with length 2 after executing mid = (end – start)/2 + start.
4) …

(2) The second problem: "Sqrt(x) [http://leetcode.com/onlinejudge] "

Implement int sqrt(int x). Compute and return the square root of *x*.

An obvious way to solve this problem is search. The square root of *x* must be in *[0, x].* So we can use binary search to find the result. Let me change this problem a little bit so that it can fits to our framework.

In array *A[0, …, x]={0, …, x}*, find *target* to make *target = sqrt(x)*.

According to the definition of sqrt(x), the invariant relation for this problem is : *A[start] <= target(target is sqrt(x)) < A[end]*. **I emphasize once again, the invariant relation guides us coding.**

a. Initialization

Start = -1(It must be -1, no other choices) and *end = x + 1*. At this

time, I'll not say *end* must be *x+1*. Actually *end* can be *x+2, x+3, ...* as long as no overflow. Think about why? But we put *x+1* here for efficiency.

b. Maintaining invariant relation inside the loop

Since our invariant relation is *A[start] <= target < A[end]* now, using ***A[mid] > target*** to maintain it.

c. Return

Based on the invariant relation, the return should be *start* for this problem.

d. Halting proof

The same inductive reference.

3. Summarization

Invariant relation leads us to the success not only for the binary search problem but also for other coding problem. Finding the invariant relation of the problem, and then everything becomes easy.

4. Exercise

Solving search for an range [http://leetcode.com/onlinejudge] in leetocde.

5. Reference

(1) http://discuss.leetcode.com/questions/213/search-for-a-range [http://discuss.leetcode.com/questions/213/search-for-a-range]
(2) http://discuss.leetcode.com/questions/214/search-insert-position [http://discuss.leetcode.com/questions/214/search-insert-position]
(3) Programming Pearls

6. Appendix

(1). Source code for "search insert position [http://leetcode.com/onlinejudge] ":

```
class Solution {
public:
```

```
    int searchInsert(int A[], int n, int target) {
        // Start typing your C/C++ solution below
        // DO NOT write int main() function
        int start = -1;
        int end = n;

        while (end - start > 1) {
            int mid = (end - start) / 2 + start;
            if (A[mid] >= target) {
                end = mid;
            }
            else {
                start = mid;
            }
        }

        return end;
    }
};
```

(2) For "sqrt(x) [http://leetcode.com/onlinejudge] "

```
[http://www.blogger.com/blogger.g?blogID=3437517718391875085] class
[http://www.blogger.com/blogger.g?blogID=3437517718391875085] Solution {
public:
    int sqrt(int x) {
        // Start typing your C/C++ solution below
        // DO NOT write int main() function
        long long int start = -1;
        long long int end = (long long int)x+1;

        while (end - start > 1) {
            long long int mid = (end - start) / 2 + start;
            if (mid*mid > x) {
                end = mid;
            } else {
                start = mid;
            }
        }
```

```
            return start;
        }
    };
```

(3) For “search for a range [http://leetcode.com/onlinejudge] ”

```
class Solution {
private:
    int searchTheFirst(int A[], int start, int end, int target) {
        while (end - start > 1) {
            int mid = (end - start) / 2 + start;
            if (A[mid] < target) {
                start = mid;
            }
            else {
                end = mid;
            }
        }

        return A[end] == target ? end : -1;
    }

    int searchTheLast(int A[], int start, int end, int target) {
        while (end - start > 1) {
            int mid = (end - start) / 2 + start;
            if (A[mid] <= target) {
                start = mid;
            }
            else {
                end = mid;
            }
        }

        return A[start] == target ? start : -1;
    }

    int searchTheFirst(int A[], int n, int target) {
        int left = -1;
```

```
        int right = n;
        return searchTheFirst(A, left, right, target);
    }

    int searchTheLast(int A[], int n, int target)  {
        int left = -1;
        int right = n;
        return searchTheLast(A, left, right, target);
    }

public:
    vector<int> searchRange(int A[], int n, int target) {
        // Start typing your C/C++ solution below
        // DO NOT write int main() function
        vector<int> result(2, -1);

        int leftIndex = searchTheFirst(A, n, target);
        if (leftIndex == -1) {
            return result;
        }
        result[0] = leftIndex;

        int rightIndex = searchTheLast(A, leftIndex, n, target);
        result[1] = rightIndex;

        return result;
    }
};
```

Posted 13th July 2013 by FihopZz

Labels: leetcode; binary search

10 View comments

Anonymous 7/17/2013 2:14 AM

Did you try your template with "Search in Rotated Sorted Array"?

Reply

Anonymous 7/17/2013 2:15 AM

Did you try your template with "Search in Rotated Sorted Array"?

Reply

Anonymous 7/17/2013 2:15 AM

Did you try your template with "Search in Rotated Sorted Array"?

Reply

Anonymous 4/27/2014 9:45 AM

Hello, the post is very useful for binary search implementation. Could you explain why the invariant relation in sqrt(x) is A[start] <= target(target is sqrt(x)) < A[end]? I think A[start] < target(target is sqrt(x)) <= A[end] should also work fine here.

Reply

Replies

Anonymous 10/26/2014 1:47 AM

The binary search implementation is useful for what it does, locate an insertion index. As the role of a traditional binary search, it fails. A binary search should return -1 if the index is not located. With the function as implemented, you can never tell if the value was actually found in the list or not since it always returns a positive index value.

Anonymous 4/15/2015 4:57 PM

But for sqrt, you can always find a non-negative index value, right?

Reply

Zewen Peng 5/05/2014 12:10 AM

I wrote code with you template, but it didn't works for the problem Sqrt(X) and "search for range". For the probelm "search for range", it shows error when the test case is A[] = {2,2} target is 3.

Reply

PTMF2013 8/25/2014 8:46 PM

This comment has been removed by the author.

Reply

PTMF2013 8/25/2014 8:51 PM

Thanks for your great conclusion!

Here is my framework for "search range", it also works and is concise, and I think start from 0, end with n is more reasonable :)

```
class Solution {
public:
vector searchRange(int A[], int n, int target) {
int leftIdx = lower_bound(A, n, target);
if (leftIdx == n || A[leftIdx] != target) return vector{-1, -1}; // not found
int rightIdx = upper_bound(A, n, target);
return vector{leftIdx, rightIdx};
}

int lower_bound(int A[], int n, int target) {
int left = 0, right = n - 1;
while (left <= right) {
int mid = left + (right-left)/2;
if (target <= A[mid]) right = mid - 1;
```

```
else left = mid + 1;
}
return left;
}

int upper_bound(int A[], int n, int target) {
int left = 0, right = n - 1;
while (left <= right) {
int mid = left + (right-left)/2;
if (target < A[mid]) right = mid - 1;
else left = mid + 1;
}
return right;
}
};
```

Reply

Vivian Yang 4/17/2015 1:21 PM

I think for sqrt(x) the start should be 0? cause target >= start and start won't be -1 anyway

Reply