CSC236 fall 2018

correct after & before

Danny Heap
heap@cs.toronto.edu / BA4270 (behind elevators)
http://www.teach.cs.toronto.edu/~heap/236/F18/
416-978-5899

Using Introduction to the Theory of Computation, Chapter 2

Outline

iterative binary search

power

notes

correctness by design

same binary search algorithm, but now using iteration (loop)

draw pictures of before, during, after

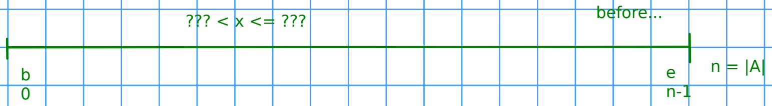
pre: A sorted, comparable with x

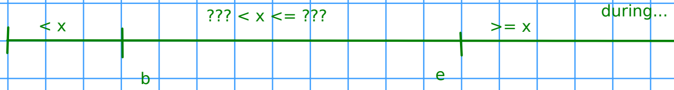
post: $0 \leq b \leq n$ and A[0:b] $< x \leq$A[b:n-1]

may be empty

may be empty

draw the state of A before, during, and after loop executes ---------------------------->





examination of three pictures leads to
draft loop:

```
while b <= e:
    m = (b + e)//2
    if A[m] < x:
        b = m + 1  # want A[b-1] < x
    else: # A[m] >= x
        e = m - 1    # want A[e+1] >= x
return b  # see "after" picture
```

“derive” conditions from pictures

need notation for mutation

e at the end of ith loop iteration: e_i
b at the end of ith loop iteration: b_i
(no need to subscript variables that don't change in loop)
We will treat names as though introduced by the code...

Precondition: A is sorted array of elements comparable to x, |A| = n > 0, 0=b<= e= n-1
Postcondition: 0 <= b <= n AND all([j<x for j in A[0:b]) AND all([k>=x for k in A[e+1:n]) AND e, b \in \N

We want loop invariant P(i) to be (a) true after every loop interation (including the last one) and (b) to lead toward Postcondition.

define P(i): At the end of loop iteration i (if it occurs)
0 <= b_i <= e_i + 1 <= n AND b_i, e_i+1 \in \N AND all([j<x for j in A[0:b]])
AND all([k>=x for k in A[e+1:n]])
Prove \forall i \in \N, P(i), using simple induction on i.

base case i=0: Since the loop has not iterated even once, b_i = 0, e_i = n-1, so 0<=b_i<=e_i+1=n, and
b_i=0, e_i non-neg integer are both in \N. Since A[0:b] and A[e+1:n] are both empty slices,
any universally quantified claim about their elements is vacuously true. So P(0) holds.

inductive step: Let i \in \N, and assume P(i). I will show that P(i+1) follows. If there is an (i+1)th iteration of the
loop then (by the loop condition) b_i <= e_i, so (by the next line of code) m = (b_i + e_i)//2.
Note that this means that b_i = 2b_i//2 <= m <= 2e_i/2 = e_i.
Also m \in \N, since it is the sum of naturals (hence natural) integer divided by 2 (hence natural).

case A[m] < x:
By the code b_{i+1} = m+1 and e_{i+1} = e_i, so 0<=b_i<=m+1<=e_{i+1}+1<=n
since from IH 0<=b_i AND m>=b_i AND m<=e_i AND e_i+1<=n
By the IH all([k>=x for k in A[e_i+1:n]]) so all([k>=x for k in A[e_{e+1}+1:n]]) # e_i=e_{i+1}
Also b_{i+1} = m+1, so b_{i+1} - 1 = m and A[m] < x.
Since A is sorted all([k<x for k in])
By the IH e_i+1 = e_{i+1}+1 \in \N, and b_i \in \N,
so b_{i+1} = m+1 \in \N, sum of m and 1.

Computer Science
UNIVERSITY OF TORONTO

“derive” conditions from pictures

need notation for mutation

```
case A[m] >= x:
      By code b_{i+1} = b_i and e_{i+1} = m-1, so 0<=b_{i+1}<=e_{i+1}+1<=n
                                                       # by IH 0<=b_i=b_{i+1}
                                                       # AND b_i<=m=e_{i+1}+1
                                                       # AND e_{i+1}+1=m<=e_i+1<=n (by IH)
      so by IH all([j<x for j in A[0:b_i]]) ==> all([j<x for j in A[0:b_{i+1}]])
      Also, A[m] = A[e_{i+1}+1] >= x and A is sorted, so all([k>=x for k in A[e_{i+1}+1:n]])
      Also b_i=b_{i+1} \in \N, by IH and e_{i+1} = m - 1 is an integer >= -1 (since m is \in \N),
                                            so e_{i+1} + 1\in \N.
```

partial correctness

precondition+execution+termination imply postcondition

loop invariant helps get us closer

if the loop terminates at, say, iteration f, we have:

```
b_f > e_f                 # since the loop condition is violated
AND b_f  - 1 <= e_f  # by P(f), loop invariant...
Since b_f, e_f are integers, this  means b_f = e_f + 1, so we can replace e_f + 1 in
P(f): 0<=b_f<=n AND all([j<x for j in A[b_f:n]) AND all([k>=x for k in A[b_f:n]])
```

This is the postcondition.

It only remains to prove that the loop terminates...--->

do we have termination?

It is generally difficult to reason precisely that "eventually" the loop condition is violated. Don't *ever* do that! The successful approach is to identify some expression based on the variables that is (a) decreasing with each loop iteration and (b) a natural number. You then conclude that a decreasing sequence of natural numbers is finite (it has a least element!), and hence finite. Since it's finite there must be a last loop iteration...

A good choice here is $e_i + 1 - b_i$ (basically the length of the portion of A that is being searched). We need to prove that it gets smaller with each loop iteration, that is if there is an (i+1)th iteration, then $e_i + 1 - b_i > e_{i+1} + 1 - b_{i+1}$. There are just two cases to consider, and we use the inequality derived earlier that $b_i <= m <= e_i$ when $b_i <= e_i$.

case $A[m] < x$, so $e_{i+1} = e_i$ AND $b_{i+1} = m+1$. Then

$$e_{i+1} + 1 - b_{i+1} = e_i + 1 - (m+1) = e_i - m < e_i + 1 - m <= e_i + 1 - b_i \quad \text{\# since } b_i <= m$$

case $A[m] > x$, so $e_{i+1} = m - 1$ AND $b_{i+1} = b_i$. Then

$$e_{i+1} + 1 - b_{i+1} = m - 1 + 1 - b_i = m - b_i < m + 1 - b_i <= e_i + 1 - b_i \quad \text{\# since } m <= e_i$$

In both cases the expression is decreasing. By the loop invariant we know that $b_i \in \mathbb{N}$ and $e_i + 1 \in \mathbb{N}$, so we have exhibited a decreasing sequence in $\mathbb{N}$. Such a sequence is finite, so the corresponding loop iterations are also finite. QED

correctness by discovery

integer power

```
def power(x, y) :
  z = 1
  m = 0
  while m < y :
    z = z * x
    m = m + 1
  return z
```

- precondition?
- postcondition?
- notation for mutation

partial correctness

precondition+execution+termination imply postcondition

a loop invariant helps get us closer

partial correctness

precondition+execution+termination imply postcondition

a loop invariant helps get us closer

prove partial correctness

prove termination

associate a decreasing sequence in $\mathbb{N}$ with loop iterations

it helps to add claims to the loop invariant

put it together — correctness

notes