Binary search

Write a subroutine

proc search(t : int, x : seq $\langle\mathbf{int}\rangle$) **returns** $result$: int ...

that returns

- if t occurs in array x, a number p such that $x(p) = t$
- if t does not occur in array x, the number -1

Assume that x is (or points to) an array of length ≥ 0, sorted in nondecreasing order

$$x(0) \leq x(1) \leq \cdots \leq x(x.\text{length} - 1)$$

Since x won't change we will take this as "background knowledge" rather than repeating it in all the assertions.

Running time should be about proportional to $\log_2(x.\text{length})$.

Specification

First let's introduce some useful notation

- $x\,S = \{x(i) \mid i \in S\}$ where S is a set of indices
- In particular $x\{p, ..r\} = \{x(i) \mid p \leq i < r\}$

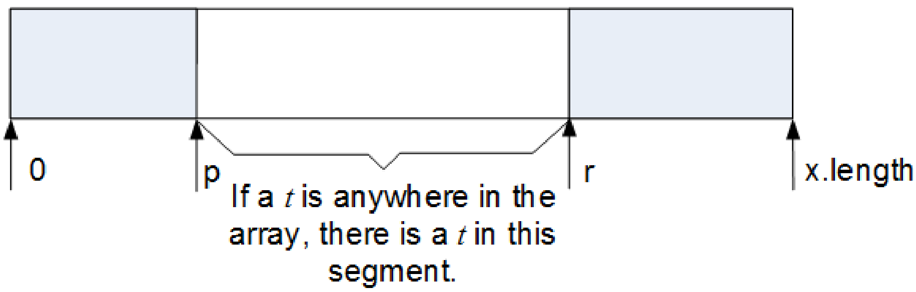
Now we can give a postcondition using $result$ to represent the value returned

$$\mathcal{R} : \begin{array}{ll} & (t \in x\{0, ..x.\text{length}\} \Rightarrow x(result) = t) \\ \wedge & (t \notin x\{0, ..x.\text{length}\} \Rightarrow result = -1) \end{array}$$

Invariant

Idea: Try to trap a t in a region of the array between two "indices".

Invariant 0: If a t is in the array at all, there is one in the region $\{p, ..r\}$



More formally

Invariant 0: $t \in x\{0, ..x.\text{length}\} \Rightarrow t \in x\{p, ..r\}$

To be sure that invariant 0 makes sense, we should require that p and r are "in range".

Invariant 1: $0 \le p \le r \le x.\text{length}$

We have

$$\{\ \text{true}\ \}$$
$$?$$
$$\left\{ \mathcal{I} : \begin{array}{rl} & 0 \le p \le r \le x.\text{length} \\ \wedge & (t \in x\{0, ..x.\text{length}\} \Rightarrow t \in x\{p, ..r\}) \end{array} \right\}$$
$$?$$
$$\{\ R\ \}$$

Initialization

It is easy to establish this invariant in the first place

$\{\text{ true }\}$
$p := 0$
$r := x.\text{length}$
$$\left\{ \mathcal{I} : \begin{array}{cc} & 0 \le p \le r \le x.\text{length} \\ \wedge & (t \in x\,\{0, ..x.\text{length}\} \Rightarrow t \in x\,\{p, ..r\}) \end{array} \right\}$$
?
$\{\ \mathcal{R}\ \}$

We must check that $\mathcal{I}[r : x.\text{ length}][p : 0]$ is universally true.

Iteration

Use $\mathcal{I}$ as a loop invariant

$\{\text{ true }\}$
$p := 0$
$r := x.\text{length}$
$$\left\{ \mathcal{I} : \begin{array}{cc} & 0 \le p \le r \le x.\text{length} \\ \wedge & (t \in x\,\{0, ..x.\text{length}\} \Rightarrow t \in x\,\{p, ..r\}) \end{array} \right\}$$
while $\mathcal{G}$ do
  $\{\ \mathcal{I} \wedge \mathcal{G}\ \}$
  ?b
  $\{\ \mathcal{I}\ \}$
end while
$\{\mathcal{I} \wedge \neg\mathcal{G}\}$
?
$\{\ \mathcal{R}\ \}$

When should we stop?

Loop guard

When the size of the interval ($r - p$) is 1 or 0, we can no longer split it into disjoint, nonempty subsets.

So use $r - p < 2$ as $\neg\mathcal{G}$.

$p := 0$

$r := x.\mathrm{length}$

$$\left\{ \mathcal{I} : \begin{array}{ll} & 0 \le p \le r \le x.\mathrm{length} \\ \wedge & (t \in x\,\{0, ..x.\mathrm{length}\} \Rightarrow t \in x\,\{p, ..r\}) \end{array} \right\}$$

while $\mathcal{G} : r - p \ge 2$ do

{ $\mathcal{I} \wedge \mathcal{G}$ }

?b

{ $\mathcal{I}$ }

end while

{ $\mathcal{I} \wedge \neg\mathcal{G}$ }

?

{ $\mathcal{R}$ }

Calculating the result

$$\left\{\begin{array}{l} \quad \mathcal{I} : \left(\begin{array}{l} \quad 0 \le p \le r \le x.\text{length} \\ \wedge \ (t \in x\,\{0, ..x.\text{length}\} \Rightarrow t \in x\,\{p, ..r\}) \end{array}\right) \\ \wedge \ r - p < 2 \end{array}\right\}$$

?

$$\left\{\mathcal{R} : \begin{array}{l} \quad (t \in x\{0, ..x.\text{length}\} \Rightarrow x(\mathit{result}) = t) \\ \wedge \ \ (t \notin x\{0, ..x.\text{length}\} \Rightarrow \mathit{result} = -1) \end{array}\right\}$$

We find the last command as follows:

- If $p = r$ then $\{p, ..r\} = \emptyset$ and so $t \in x\,\{p, ..r\}$ is false and, from the invariant, that means that $t \in x\,\{0, ..x.\text{length}\}$ is also false. Thus -1 is the appropriate result
- On the other hand, if $p \neq r$, then $r = p + 1$ and, from the invariant,

  $$0 \le p < p + 1 = r \le x.\text{length}$$

  So p is a legitimate index of x.
  - ∗ Of course, if $t = x(p)$, then p is an acceptable result.
  - ∗ Now $t \in x\,\{p, ..r\}$ simplifies to $t = x(p)$ and, if this is false, then, from the invariant, $t \in x(\{0, ..x.\text{length}\}$ is also false and -1 is the correct result.

So we have

$\{\ \mathcal{I} \wedge r - p < 2\}$
if $p = r$ then $\mathit{result} := -1$
elsif $t = x(p)$ then $\mathit{result} := p$
else $\mathit{result} := -1$ end if
$\{\ \mathcal{R}\ \}$

Loop body

Now it remains to implement the loop body ?b which

- Needs to preserve the invariant
- And should bring the loop "closer to termination"

Our remaining problem

$\{ \mathcal{I} \wedge r - p \geq 2 \}$?b $\{ \mathcal{I} \}$

Combining $0 \leq p \leq r \leq x.\text{length}$ with $r - p \geq 2$ we have

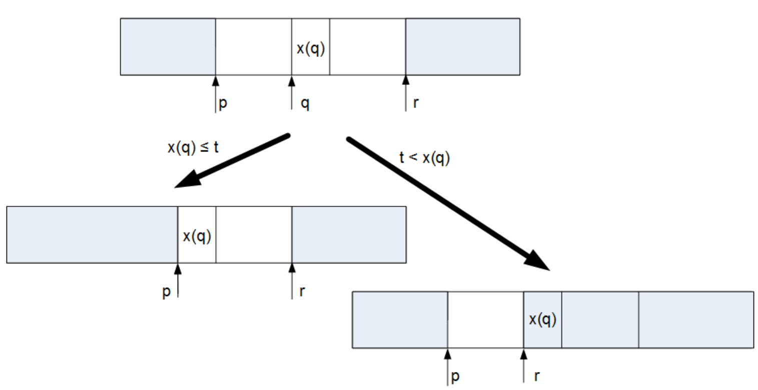
$$0 \leq p < p + 1 < r \leq x.\text{length}$$

If q is such that $p < q < r$, the interval $\{p, ..r\}$ can be split into two nonempty and disjoint intervals $\{p, ..q\}$ and $\{q, ..r\}$.

We know from $\mathcal{I}$ that if t is anywhere in the array, it is in $x\{p, ..r\}$

Because the array is sorted:

- if $x(q) \leq t$ then, if a t is anywhere in the array, one must be in $x\{q, ..r\}$
- if $x(q) > t$ then, if a t is anywhere in the array, one must be in $x\{p, ..q\}$



Loop body (continued)

So this leads to

$\{ \mathcal{I} \wedge r - p \geq 2 \}$
$?$
$\{ \mathcal{I} \wedge p < q < r \}$
if $x(q) \leq t$ then $p := q$ else $r := q$ end if
$\{ \mathcal{I} \}$

What about "should bring the loop 'closer to termination'"?

- Since $p < q < r$, the value of $r - p$ gets smaller with each iteration of the loop and so must eventually become smaller than 2.

Choosing q:

- Any q such that $p < q < r$ will do.
- For efficiency, we want to make $\{p, ..q\}$ and $\{q, ..r\}$ roughly the same size.
- So a good choice is $q := \left\lfloor \frac{p+r}{2} \right\rfloor$.

In summary

proc search(t : int, x : seq $\langle\mathbf{int}\rangle$) returns $result$: int

requires x is sorted in nondecreasing order

$$x(p) \le x(p+1) \le \cdots \le x(x.\text{length} - 1)$$

ensures

$$(t \in x\{0, ..x.\text{length}\} \Rightarrow x(result) = t)$$
$$\wedge (t \notin x\{0, ..x.\text{length}\} \Rightarrow result = -1)$$

var $p := 0$

var $r := x.\text{length}$

$$\left\{ \mathcal{I} : \begin{array}{ll} & 0 \le p \le r \le x.\text{length} \\ \wedge & (t \in x(\{0, ..x.\text{length}\}) \Rightarrow t \in x\,\{p, ..r\}) \end{array} \right\}$$

while $\mathcal{G} : r - p \ge 2$ do

$\{\ \mathcal{I} \wedge \mathcal{G}\ \}$

val $q := \left\lfloor \frac{p+r}{2} \right\rfloor$

$\{\mathcal{I} \wedge \mathcal{G} \wedge p < q < r\}$

if $x(q) \le t$ then

$p := q$

else

$r := q$

end if

end while

$\{\ \mathcal{I} \wedge r - p < 2\}$

if $p = r$ then $result := -1$

elsif $t = x(p)$ then $result := p$

else $result := -1$ end if

end search

In Java

```
/** Do a binary search for t in array x
 * <p>Precondition: x is sorted in nondecreasing order
 * <p>Changes: nothing
 * <p>Postconditions: <ul>
 * <li>  if t occurs in x, then the result is such that x[result]==t
 * <li>  if t does not occur in x, then the result is -1
 * </ul>  */
public static int search( int t, int[] x ) {
    int p = 0 ;
    int r = x.length ;
    //invariant: 0 <= p && p <= r && r <= x.length
    //invariant: if t occurs in x, it occurs in x[{p,..r}]
    while( r-p >= 2 ) {
        int q = p+(r-p)/2 ;
        // p < q < r
        if( x[q] <= t ) p = q ;
        else r = q ; }
    if( p==r ) return -1 ;
    else if( x[p]==t ) return p ;
    else return -1 ; }
```

Time

Each iteration cuts the size of $r - p$ roughly in two, thus the number of iterations is approximately $\log_2(x.\mathrm{length})$.

For example if $x.\mathrm{length} = 129$, the worst-case values for $r - p$ are

$$129, 65, 33, 17, 9, 5, 3, 2, 1$$

so 8 iterations before $r - p < 2$. While $\lceil \log_2 129 \rceil = 8$.

In fact, the number of iterations is either $\lfloor \log_2(x.\mathrm{length}) \rfloor$ or $\lceil \log_2(x.\mathrm{length}) \rceil$.