CS1110 4 November 2008
Developing array algorithms. Reading: 8.5

Important point: how we create the invariant, as a picture

Haikus (5-7-5) seen on Japanese computer monitors

Yesterday it worked.
Today it is not working.
Windows is like that.

A crash reduces
Your expensive computer
To a simple stone.

Three things are certain:
Death, taxes, and lost data.
Guess which has occurred?

Serious error.
All shortcuts have disappeared.
Screen. Mind. Both are blank.

The Web site you seek
Cannot be located, but
Countless more exist.

Chaos reigns within.
Reflect, repent, and reboot.
Order shall return.

Developing algorithms on arrays

We develop several important algorithms on arrays.

With each, we specify the algorithm by giving its precondition and postcondition as pictures.

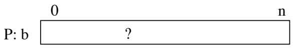
Then, draw the invariant by drawing another picture that “generalizes” the precondition and postcondition, since the invariant is true at the beginning and at the end.

Four loopy questions —memorize them:

1. How does loop start (how to make the invariant true)?
2. When does it stop (when is the postcondition true)?
3. How does repetend make progress toward termination?
4. How does repetend keep the invariant true?

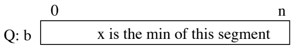
Invariant as picture: Combining pre- and post-condition

Finding the minimum of an array. Given array b satisfying precondition P, store a value in x to truthify postcondition Q:



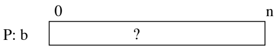
and n >= 0

(values in 0..n-1 are unknown)



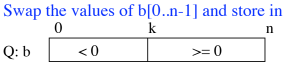
The invariant as picture: Combining pre- and post-condition

Put negative values before nonnegative ones. given precondition P:



(values in 0..n-1 are unknown)

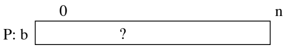
Swap the values of b[0..n-1] and store in k to truthify Q:



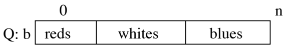
(values in 0..k-1 are < 0,
values in k..n-1 are > 0)

The invariant as picture: Combining pre- and post-condition

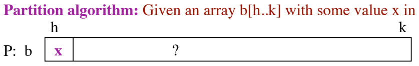
Dutch national flag. Swap values of 0..n-1 to put the reds first, then the whites, then the blues. That is, given precondition P, swap value of b[0.n] to truthify postcondition Q:



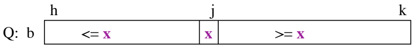
(values in 0..n-1 are unknown)

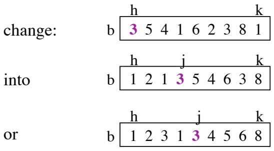


Partition algorithm: Given an array b[h..k] with some value x in b[h]:



Swap elements of b[h..k] and store in j to truthify P:





x is called the pivot value.

x is not a program variable; x just denotes the value initially in b[h].

Linear search (for value known to be in array)

Vague spec.: Find first occurrence of v in b[h..k-1], which is known to be there.
Better spec.: Store an integer in i to truthify postcondition Q:

P: 1. v is not in b[h..i-1]
2. v = b[k]

```
                     h                           k
precondition P: b   |        v is in here        |

                     h            i              k
postcondition Q: b  | v not here |v|     ?       |
```

Can't simply combine P and Q because position of v not known initially. But we can just delete value v from Q:

```
                     h            i              k
    invariant Q: b  | v not here |       ?       |
```

Linear search

Vague spec.: Find first occurrence of v in b[h..k-1].
Better spec.: Store an integer in i to truthify postcondition Q:

P: 1. v is not in b[h..i-1]
2. i = k OR v = b[k]

```
        h                             k
P: b   |          v is in here        |

        h              i              k
Q: b   |   x not here |x|     ?       |

                                      i
        h                             k
Q: b   |            x not here        |
```

Binary search: Vague spec: Look for v in **sorted** array segment b[h..k].
Better spec:
Precondition P: b[h..k] is sorted (in ascending order).
Store in i to truthify:
Postcondition Q: b[h..i] <= v and v < b[i+1..k]

Below, the array is in non-descending order:

```
        h                              k
P: b   |              ?                |

        h               i              k
Q: b   |      <= v     |      > v      |
```

Called binary search because each iteration of the loop cuts the array segment still to be processed in half

Reversal: Reverse the elements of array segment b[h..k].

```
                   h                                    k
precondition P:   |             not reversed            |

                   h                                    k
postcondition Q:  |               reversed              |
```

```
                h                     k
Change:     b  | 1 2 3 4 5 6 7 8 9 9 9 9 |
                h                     k
into        b  | 9 9 9 9 8 7 6 5 4 3 2 1 |
```

Remove adjacent duplicates

```
              0                         n
change:   b  | 1  2  2  4  2  2  7  8  9  9  9  9 |

              0                 h       n
into      b  | 1  2  4  2  7  8  9  8  9  9  9  9 |
```

don't care what is in b[k+1..n]

Truthify:

b[0..h] = initial values in b[0..n] but with adj dups removed

```
                      h                              k
Precondition P:  b   |               ?               |

                      h                       i      k
Postcondition Q: b   | initial values of b[0..k] | unchanged |
                     | with no duplicates        |           |
```