CS1110

Lecture 20: **Sequence algorithms**

Announcements

Upcoming schedule

Plan as of March 29, 2013, when these slides were printed:

Today (April 4): A4 due, A6 out

Tu Apr 9: lecture on searching and sorting – on the exam

Probably a new lab exercise, for prelim exercise

Th Apr 11: lecture = review session

Fri Apr 12: A6 due? (*may have changed* since March 28).

Tu Apr 16: lecture = office hours, in Thurston 102

Exam that evening, same location as before

Probably no new lab exercise that week

Slides by D. Gries, L. Lee, S. Marschner, W. White

Sorting: A Key Algorithmic Family

Q: Given a list of items, how can we arrange for them to be sorted in increasing order, in a time- and space-efficient manner?

Applications: making items easier to find.[1]

```
def sort(b, h, k):
    """Make b[h..k] sorted.  Pre: b: list of ints; k>=h-1"""
    # start with b[h], and move it somewhere...?
```

[1] Also, computing poker-hand scores.

Motivation: A Famous Sorting Function

```
def qsort(b, h, k):
    """Make b[h..k] sorted.
    Pre: b: list of ints; k>=h-1"""
```

Clicker Q2: base case

```
    i = partition(b, h, k)
```

Clicker Q1: recursive case

```
def partition(b, h, k):
    """Let x = b[h] be the pivot value.  Rearrange b[h..k] so
    that there is an i in h..i
    where  b[h..i-1] <= x, b[i]=x;
    b[i+1..k] >=x.  Return i.
    Pre: k>=h"""
    # Can you do this without
    # creating extra lists?
```

Pictorial Notation for Sequence Assertions

b	0 some property p	h some property q k

Equivalent to:

Property p holds on all items in b[0..h-1], and property q holds on all items in b[h..k].

(The precise location of the "vertical bars" matters.)

Can also indicate single items. (h h+1)

((h +1) – h = 1; it's all consistent, hurrah.)

Partition Algorithm

- Given a sequence b[h..k] with some value x in b[h]:

pre: b [h] x | ? [k]

- Swap elements of b[h..k] and store in i to truthify post:

post: b [h] <= x | [i] x | [i+1] >= x [k]

change: b 3 5 4 1 6 2 3 8 1 (h … k)

into b 1 2 1 3 5 4 6 3 8 (h, i = 3, k)

or b 1 2 3 1 3 4 5 6 8 (h, i = 3, k)

- x is called the pivot value
  - x is not a program variable, but a standin for a number: value initially in b[h]

An Invariant to Guide Our Thinking

- Given a sequence b[h..k] with some value x in b[h]:

pre: b [h] x | ? [k]

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

post: b [h] <= x | [i] x | [i+1] >= x [k]

inv: b [h] <= x | [i] x | ? [j] | >= x [k]

- Agrees with precondition when i = h, j = k+1
- Agrees with postcondition when j = i+1

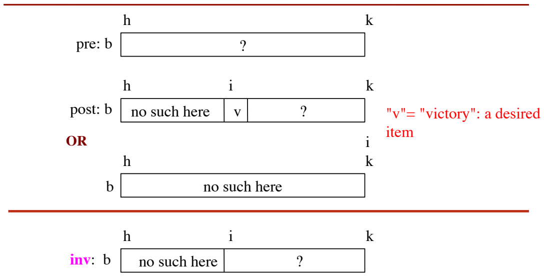
Partition Algorithm Implementation

```
def partition(b, h, k):
    """Partition list b[h..k] around a pivot x = b[h];
    Return index of pivot point. Assume a swap function _swap(b,ind1, ind2).
   Pre: k>=h"""
   CLICKER Q5
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x, b[i+1..j-1] unknown
    while CLICKER Q4
        if b[i+1] >= x:
         # Move to end of block.
         _swap(b,i+1,j-1)
         j = j - 1
      else:   # b[i+1] < x
        CLICKER Q3
    # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x
    return i
```

Developing Algorithms on Sequences

- Specify the algorithm by giving its precondition and postcondition as pictures.
- Draw the invariant by drawing another picture that "generalizes" the precondition and postcondition
  - The invariant is true at the beginning and at the end
- The four loop design questions (memorize them)
  1. How does loop start (how to make the invariant true)?
  2. How does it stop (is the postcondition true)?
  3. How does repetend make progress toward termination?
  4. How does repetend keep the invariant true?

Linear Search (Generalizable index/find)



Linear Search (Index/Find Version)

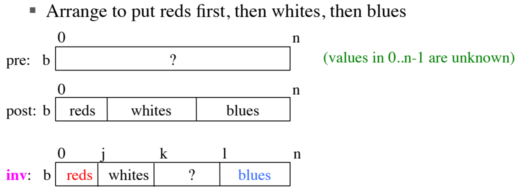
```
def linear_search(b,c,h):
    """Returns: index of 1st occurrence of c
    in b[h..], or -1 if not found"""
    # Store in i the index of the first c in b[h..]
    # what init?

    # invariant: c is not in b[0..i-1]
    while # what?
        #what?

    # post: b[i] == c and c is not in b[h..i-1]
    return i if i < len(b) else -1
```

Analyzing the Loop

1. Does the initialization make **inv** true?
2. Is **post** true when **inv** is true and **condition** is false?
3. Does the repetend make progress?
4. Does the repetend keep **inv** true?

Famous "Sort-Like" Example

- Dutch national flag: tri-color
  - Sequence of 0..n-1 of red, white, blue "pixels"
  - Arrange to put reds first, then whites, then blues



b[0..j-1] red, b[j..k-1] white, b[l..n-1] blue, b[k..l-1] unknown

Dutch National Flag Algorithm

```
def dnf(b, h, k):
    """(DNF explanation omitted for space.)
    Returns: split-points as a tuple (i,j)"""
   # init?
    # inv: b[h..t-1] < 0, b[t..i-1] ?, b[i..j] = 0, b[j+1..k] > 0
    while t < i:
       if b[i-1] < 0:
          # what?

       elif b[i-1] == 0:
          # what?
       else:
          # what?

    # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    return (i, j)
```