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
Th Apr 11: lecture = review session
Fri Apr 12: A6 due? (may have changed since March 28).
Exam that evening, same location as before

Motivation: A Famous Sorting Function

def gsort(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?

Clicker Q2: base case
Clicker Q1: recursive case

Pictorial Notation for Sequence Assertions

<table>
<thead>
<tr>
<th>b</th>
<th>0</th>
<th>h</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>some property p</td>
<td>some property q</td>
<td></td>
</tr>
</tbody>
</table>

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.

Partition Algorithm

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

  pre: b
  change: b
  post: b

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

  b
  change: b
  post: b

- 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
  change: b
  post: b

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

  b
  change: b
  post: b

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

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.

1Also, computing poker-hand scores.
**Partition Algorithm Implementation**

```python
def partition(b, h, k):
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x, b[i+1..j-1] unknown
    while
        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)**

```plaintext
<table>
<thead>
<tr>
<th>h</th>
<th>i</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre: b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>post: b no such here</td>
<td>v</td>
<td></td>
</tr>
</tbody>
</table>
```

---

**Linear Search (Index/Find Version)**

```python
def linear_search(b, c, h):
    # Store in i the index of the first c in b[h..]
    # 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
```

---

**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
- Dutch National Flag Algorithm
  ```python
  def dnf(b, h, k):
      # invariant: b[h..t-1] < 0, b[t..i-1] ?, b[i..j] = 0, b[j+1..k] > 0
      while
          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
  ```

---

**Dutch National Flag Algorithm**

- Sequence of 0..n-1 of red, white, blue "pixels"
- Arrange to put reds first, then whites, then blues