Lecture 26

Sequence Algorithms (Continued)
Announcements for This Lecture

Assignment & Lab

- A6 is not graded yet
  - Done early next week
- A7 due **Mon, Dec. 4**
  - But extensions possible
  - Just ask for one!
  - But make good effort
- Lab Today: Office Hours
  - Get help on A7 paddle
  - Anyone can go to any lab

Next Week

- Last Week of Class!
  - Finish sorting algorithms
  - Special final lecture
- Lab held, but is optional
  - Unless only have 10 labs
  - Also use lab time on A7
- Details about the exam
  - Multiple review sessions
Recall: Horizontal Notation

Example of an assertion about an sequence \( b \). It asserts that:

1. \( b[0..k-1] \) is sorted (i.e. its values are in ascending order)
2. Everything in \( b[0..k-1] \) is \( \leq \) everything in \( b[k..\text{len}(b)-1] \)

Given index \( h \) of the first element of a segment and index \( k \) of the element that follows that segment, the number of values in the segment is \( k - h \).

\( b[h .. k - 1] \) has \( k - h \) elements in it.
Partition Algorithm

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

$$
\begin{array}{c}
\text{h} \\
\text{pre: } b \\
\text{k}
\end{array}
\begin{array}{c|c|c}
\hline
\text{x} & \text{?} \\
\hline
\end{array}
$$

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

$$
\begin{array}{c|c|c|c|c|c}
\text{h} & \text{i} & \text{i+1} & \text{k} \\
\text{post: } b \\
\begin{array}{c|c|c|c|c|c}
\hline
\text{<= } x & \text{x} & \geq x \\
\hline
\end{array}
\end{array}
$$

$$
\begin{array}{c|c|c|c|c|c}
\text{h} & \text{i} & \text{j} & \text{k} \\
\text{inv: } b \\
\begin{array}{c|c|c|c|c|c}
\hline
\text{<= } x & \text{x} & \text{?} & \geq x \\
\hline
\end{array}
\end{array}
$$

• Agrees with precondition when $i = h$, $j = k+1$
• Agrees with postcondition when $j = i+1$
def partition(b, h, k):
    
    """Partition list b[h..k] around a pivot x = b[h]"""

    i = h; j = k + 1; x = b[h]
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x

    while i < j-1:
        if b[i+1] >= x:
            # Move to end of block.
            swap(b, i+1, j-1)
            j = j - 1
        else:
            # b[i+1] < x
            swap(b, i, i+1)
            i = i + 1

    # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x

    return i

partition(b, h, k), not partition(b[h:k+1])
Remember, slicing always copies the list!
We want to partition the original list
Partition Algorithm Implementation

```python
def partition(b, h, k):
    """Partition list b[h..k] around a pivot x = b[h]""
    i = h; j = k+1; x = b[h]
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x
    while i < j-1:
        if b[i+1] >= x:
            # Move to end of block.
            swap(b, i+1, j-1)
            j = j - 1
        else:
            # b[i+1] < x
            swap(b, i, i+1)
            i = i + 1
    # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x
    return i
```

<table>
<thead>
<tr>
<th>&lt;= x</th>
<th>x</th>
<th>?</th>
<th>&gt;= x</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>i</td>
<td>i+1</td>
<td>j</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>
def partition(b, h, k):
    """Partition list b[h..k] around a pivot x = b[h]"""
    i = h; j = k + 1; x = b[h]
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x
    while i < j - 1:
        if b[i+1] >= x:
            # Move to end of block.
            swap(b, i+1, j-1)
            j = j - 1
        else:
            # b[i+1] < x
            swap(b, i, i+1)
            i = i + 1
    # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x
    return i
def partition(b, h, k):
    """Partition list b[h..k] around a pivot x = b[h]""
    i = h; j = k+1; x = b[h]
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x
    while i < j-1:
        if b[i+1] >= x:
            # Move to end of block.
            swap(b,i+1,j-1)
            j = j - 1
        else:
            # b[i+1] < x
            swap(b,i,i+1)
            i = i + 1
    # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x
    return i
def partition(b, h, k):
    """Partition list b[h..k] around a pivot x = b[h]"""
    i = h; j = k+1; x = b[h]
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x
    while i < j-1:
        if b[i+1] >= x:
            # Move to end of block.
            swap(b,i+1,j-1)
            j = j - 1
        else:  # b[i+1] < x
            swap(b,i,i+1)
            i = i + 1
    # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x
    return i
Dutch National Flag Variant

- Sequence of integer values
  - ‘red’ = negatives, ‘white’ = 0, ‘blues’ = positive
  - Only rearrange part of the list, not all

pre: \begin{array}{|c|c|}
\hline
b & ? \\
\hline
\end{array}

post: \begin{array}{|c|c|c|}
\hline
b & <0 & =0 & >0 \\
\hline
\end{array}

inv: \begin{array}{|c|c|c|c|c|c|}
\hline
b & <0 & ? & =0 & >0 \\
\hline
\end{array}
Dutch National Flag Variant

- Sequence of integer values
  - ‘red’ = negatives, ‘white’ = 0, ‘blues’ = positive
  - Only rearrange part of the list, not all

<table>
<thead>
<tr>
<th>pre: b</th>
<th>h</th>
<th>?</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>post: b</td>
<td>&lt; 0</td>
<td>= 0</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>inv: b</td>
<td>h</td>
<td>t</td>
<td>i</td>
</tr>
<tr>
<td></td>
<td>&lt; 0</td>
<td>?</td>
<td>= 0</td>
</tr>
</tbody>
</table>

pre: t = h,
i = k+1,
j = k
post: t = i
def dnf(b, h, k):
    """Returns: partition points as a tuple (i,j)"""
    t = h; i = k+1, j = k;
    # 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:
            swap(b,i-1,t)
            t = t+1
        elif b[i-1] == 0:
            i = i-1
        else:
            swap(b,i-1,j)
            i = i-1; j = j-1
    # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    return (i, j)
Dutch National Flag Algorithm

```python
def dnf(b, h, k):
    
    """Returns: partition points as a tuple (i,j)"""
    
    t = h; i = k+1, j = k;
    
    # 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:
            swap(b, i-1, t)
            t = t+1
        elif b[i-1] == 0:
            i = i-1
        else:
            swap(b, i-1, j)
            i = i-1; j = j-1
    
    # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    
    return (i, j)
```

<table>
<thead>
<tr>
<th>h</th>
<th>t</th>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-2</td>
<td>3</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

11/22/16
Dutch National Flag Algorithm

```python
def dnf(b, h, k):
    """Returns: partition points as a tuple (i,j)"""
    t = h; i = k+1, j = k;
    # inv: b[h..t-1] < 0, b[t..i-1] > 0, b[i..j] = 0, b[j+1..k] > 0
    while t < i:
        if b[i-1] < 0:
            swap(b, i-1, t)
            t = t+1
        elif b[i-1] == 0:
            i = i-1
        else:
            swap(b, i-1, j)
            i = i-1; j = j-1
    # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    return (i, j)
```

<table>
<thead>
<tr>
<th>h</th>
<th>t</th>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-2</td>
<td>3</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

11/22/16    Sequences (Continued) 14
def dnf(b, h, k):
    
    """Returns: partition points as a tuple (i,j)"""
    t = h; i = k+1, j = k;
    # inv: b[h..t-1] < 0, b[t..i-1] > 0, b[i..j] = 0, b[j+1..k] > 0
    while t < i:
        if b[i-1] < 0:
            swap(b,i-1,t)
            t = t+1
        elif b[i-1] == 0:
            i = i-1
        else:
            swap(b,i-1,j)
            i = i-1; j = j-1
    # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    return (i, j)
Flag of Mauritius

- Now we have four colors!
  - Negatives: ‘red’ = odd, ‘purple’ = even
  - Positives: ‘yellow’ = odd, ‘green’ = even

```
pre: | b | ? |
```
```
post: | b | < 0 odd | < 0 even | ≥ 0 odd | ≥ 0 even |
```
```
inv: | b | < 0, o | < 0, e | ≥ 0, o | ? | ≥ 0, e |
```
Flag of Mauritius

<table>
<thead>
<tr>
<th>&lt;0, o</th>
<th>&lt;0, e</th>
<th>≥0, o</th>
<th>?</th>
<th>≥0, e</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>r</td>
<td>s</td>
<td>i</td>
<td>t</td>
</tr>
<tr>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>-4</td>
<td>7</td>
</tr>
<tr>
<td>-5</td>
<td>-6</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

One swap is not good enough
Flag of Mauritius

<table>
<thead>
<tr>
<th>&lt;0, o</th>
<th>&lt;0, e</th>
<th>≥0, o</th>
<th>?</th>
<th>≥0, e</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>r</td>
<td>s</td>
<td>i</td>
<td>t</td>
</tr>
<tr>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>-4</td>
<td>7</td>
</tr>
<tr>
<td>-5</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Need two swaps for two spaces
### Flag of Mauritius

<table>
<thead>
<tr>
<th>&lt; 0, o</th>
<th>&lt; 0, e</th>
<th>≥ 0, o</th>
<th>?</th>
<th>≥ 0, e</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>r</td>
<td>s</td>
<td>i</td>
<td>t</td>
</tr>
<tr>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>-4</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

And adjust the loop variables.
Flag of Mauritius

\[
\begin{array}{cccccc}
< 0, o & < 0, e & \geq 0, o & ? & \geq 0, e \\
h & r & s & i & t & k \\
-1 & -3 & -2 & -4 & 7 & 5 & -5 & -6 & 1 & 0 & \color{red}{2} & 4 \\
\end{array}
\]

See algorithms.py for Python code
Flag of Mauritius

<table>
<thead>
<tr>
<th>&lt;0, o</th>
<th>&lt;0, e</th>
<th>≥0, o</th>
<th>?</th>
<th>≥0, e</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>r</td>
<td>s</td>
<td>i</td>
<td>t</td>
</tr>
<tr>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>-5</td>
<td>-6</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

See algorithms.py for Python code
Linear Search

- **Vague**: Find first occurrence of $v$ in $b[h..k-1]$. 

11/22/16
Sequences (Continued)
Linear Search

• **Vague:** Find first occurrence of $v$ in $b[h..k-1]$.

• **Better:** Store an integer in $i$ to truthify result condition post:
  
  post:  
  1. $v$ is not in $b[h..i-1]$
  2. $i = k$ OR $v = b[i]$

11/22/16
Sequences (Continued)
Linear Search

- **Vague:** Find first occurrence of \( v \) in \( b[h..k-1] \).
- **Better:** Store an integer in \( i \) to truthify result condition post:
  
  post:
  1. \( v \) is not in \( b[h..i-1] \)
  2. \( i = k \) OR \( v = b[i] \)

\[
\begin{array}{cccc}
\hline
h & \_ & \_ & k \\
pre: b & & ? & \\
\hline
\hline
h & i & \_ & k \\
post: b & v \text{ not here} & v & ? \\
\end{array}
\]
Linear Search

- **Vague**: Find first occurrence of $v$ in $b[h..k-1]$.
- **Better**: Store an integer in $i$ to truthify result condition post:
  
  **post**:  
  1. $v$ is not in $b[h..i-1]$
  2. $i = k$ OR $v = b[i]$

\[
\begin{array}{c}
\text{h} \\
\text{pre: b} \\
\hline
\text{i} \\
\text{v not here} \\
\text{v} \\
\text{?} \\
\hline
\text{k}
\end{array}
\]

\[
\begin{array}{c}
\text{h} \\
\text{i} \\
\text{k} \\
\text{post: b} \\
\text{v not here} \\
\text{v} \\
\text{?} \\
\hline
\text{OR}
\end{array}
\]

\[
\begin{array}{c}
\text{h} \\
\text{b} \\
\text{v not here} \\
\text{k}
\end{array}
\]
Linear Search

pre: b

post: b

OR

inv: b
def linear_search(b,v,h,k):
    """Returns: first occurrence of v in b[h..k-1]""
    # Store in i index of the first v in b[h..k-1]
    i = h

    # invariant: v is not in b[0..i-1]
    while i < k and b[i] != v:
        i = i + 1

    # post: v is not in b[h..i-1]
    #       i >= k or b[i] == v
    return i if i < k else -1
Binary Search

- **Vague**: Look for \( v \) in sorted sequence segment \( b[h..k] \).
Binary Search

- **Vague:** Look for $v$ in **sorted** sequence segment $b[h..k]$.

- **Better:**
  - **Precondition:** $b[h..k-1]$ is sorted (in ascending order).
  - **Postcondition:** $b[h..i] \leq v$ and $v < b[i+1..k-1]$

- Below, the array is in non-descending order:

  \[
  \begin{array}{c|c|c|c}
  h & ? & k \\
  \hline
  \text{pre: } b & & \\
  \hline
  h & i & k \\
  \text{post: } b & \leq v & > v \\
  \end{array}
  \]
Binary Search

- **Vague:** Look for $v$ in sorted sequence segment $b[h..k]$.
- **Better:**
  - **Precondition:** $b[h..k-1]$ is sorted (in ascending order).
  - **Postcondition:** $b[h..i] \leq v$ and $v < b[i+1..k-1]$
- Below, the array is in non-descending order:

  ![Array diagram]

  Called binary search because each iteration of the loop cuts the array segment still to be processed in half.
Extras Not Covered in Class
Loaded Dice

- Sequence $p$ of length $n$ represents $n$-sided die
  - Contents of $p$ sum to 1
  - $p[k]$ is probability die rolls the number $k$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
</tr>
</tbody>
</table>

- Goal: Want to “roll the die”
  - Generate random number $r$ between 0 and 1
  - Pick $p[i]$ such that $p[i-1] < r \leq p[i]$

<table>
<thead>
<tr>
<th></th>
<th>0.1</th>
<th>0.1</th>
<th>0.1</th>
<th>0.1</th>
<th>0.3</th>
<th>0.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r$</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>0.4</td>
<td>0.7</td>
<td>1.0</td>
</tr>
</tbody>
</table>

weighted d6, favoring 5, 6
**Loaded Dice**

- **Want:** Value $i$ such that $p[i-1] < r \leq p[i]$

  \[
  \begin{array}{c|c}
  \text{pre: } b & ? \\
  \hline
  0 & \text{?} & n \\
  \end{array}
  \]

  \[
  \begin{array}{c|c|c}
  \text{post: } b & r > \text{sum} & r \leq \text{sum} \\
  \hline
  0 & i & n \\
  \end{array}
  \]

- **Same as precondition if $i = 0$**
- **Postcondition is invariant + false loop condition**
def roll(p):
    """Returns: randint in 0..len(p)-1; i returned with prob. p[i]
    Precondition: p list of positive floats that sum to 1."""
    r = random.random()  # r in [0,1)
    # Think of interval [0,1] divided into segments of size p[i]
    # Store into i the segment number in which r falls.
    i = 0;  sum_of = p[0]
    # inv: r >= sum of p[0] .. p[i-1]; pEnd = sum of p[0] .. p[i]
    while r >= sum_of:
        sum_of = sum_of + p[i+1]
        i = i + 1
    # post: sum of p[0] .. p[i-1] <= r < sum of p[0] .. p[i]
    return i

Analyzing the Loop

1. Does the initialization make \texttt{inv} true?
2. Is \texttt{post} true when \texttt{inv} is true and \texttt{condition} is false?
3. Does the repetend make progress?
4. Does the repetend keep \texttt{inv} true?
### Reversing a Sequence

<table>
<thead>
<tr>
<th>pre:</th>
<th>b</th>
<th>h</th>
<th>not reversed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>post:</td>
<td>b</td>
<td>h</td>
<td>reversed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>change:</th>
<th>b</th>
<th>1 2 3 4 5 6 7 8 9 9 9 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>into</td>
<td>b</td>
<td>9 9 9 9 8 7 6 5 4 3 2 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>inv:</th>
<th>b</th>
<th>h</th>
<th>i</th>
<th>j</th>
<th>not reversed</th>
<th>swapped</th>
</tr>
</thead>
</table>

11/22/16