61A Lecture 20
Announcements
Sets
Sets

One more built-in Python container type

- Set literals are enclosed in braces
- Duplicate elements are removed on construction
- Sets have arbitrary order, just like dictionary entries

```python
>>> s = {3, 2, 1, 4, 4}
>>> s
{1, 2, 3, 4}
>>> 3 in s
True
>>> len(s)
4
>>> s.union({1, 5})
{1, 2, 3, 4, 5}
>>> s.intersection({6, 5, 4, 3})
{3, 4}
>>> s
{1, 2, 3, 4}  # (Demo)
```
Implementing Sets

What we should be able to do with a set:

- **Membership testing**: Is a value an element of a set?
- **Union**: Return a set with all elements in set1 or set2
- **Intersection**: Return a set with any elements in set1 and set2
- **Adjoin**: Return a set with all elements in s and a value v
Sets as Linked Lists
Sets as Unordered Sequences

Proposal 1: A set is represented by a linked list that contains no duplicate items.

```python
def empty(s):
    return s is Link.empty

def contains(s, v):
    """Return whether set s contains value v."

>>> s = Link(1, Link(3, Link(2)))
>>> contains(s, 2)
True
```

Time order of growth

- \( \Theta(1) \)
- \( \Theta(n) \)

Time depends on whether \& where \( v \) appears in \( s \)

Assuming \( v \) either does not appear in \( s \) or appears in a uniformly distributed random location
Sets as Unordered Sequences

```python
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)

def intersect(set1, set2):
    in_set2 = lambda v: contains(set2, v)
    return filter_link(in_set2, set1)

def union(set1, set2):
    not_in_set2 = lambda v: not contains(set2, v)
    set1_not_set2 = filter_link(not_in_set2, set1)
    return extend_link(set1_not_set2, set2)
```

- **Time order of growth**
  - $\Theta(n)$
  - The size of the set
  - $\Theta(n^2)$
  - If sets are the same size
  - $\Theta(n^2)$

Return elements $x$ for which $\text{in\_set2}(x)$ returns a true value

Return a linked list containing all elements in $\text{set1\_not\_set2}$ followed by all elements in $\text{set2}$
Sets as Ordered Linked Lists
Sets as Ordered Sequences

Proposal 2: A set is represented by a linked list with unique elements that is ordered from least to greatest

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Assume that sets are...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use sets to contain values</td>
<td>Unordered collections</td>
<td>empty, contains, adjoin, intersect, union</td>
</tr>
<tr>
<td>Implement set operations</td>
<td>Ordered linked lists</td>
<td>first, rest, &lt;, &gt;, ==</td>
</tr>
</tbody>
</table>

Different parts of a program may make different assumptions about data
Searching an Ordered List

```python
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>adjoin</td>
<td>$\Theta(n)$</td>
</tr>
</tbody>
</table>

---

(Demo)
Set Mutation
Adding to an Ordered List

<table>
<thead>
<tr>
<th>Link instance</th>
<th>Link instance</th>
<th>Link instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>first: 1</td>
<td>first: 3</td>
<td>first: 5</td>
</tr>
<tr>
<td>rest:</td>
<td>rest:</td>
<td>rest:</td>
</tr>
</tbody>
</table>

`s:`

```
add(s, 0)
```
Adding to an Ordered List

```
Link instance
first: 0
rest: 

Link instance
first: 3
rest: 

Link instance
first: 5
rest: 

s:
```

```
add(s, 3)
add(s, 4)
```
Adding to an Ordered List

\[ \text{add}(s, 6) \]
Adding to an Ordered List

```
s: Link instance
  first: 0
  rest:  Link instance
    first: 3
    rest:  Link instance
      first: 4
      rest:  Link instance
        first: 5
        rest:  Link instance
          first: 6
          rest:  
```
def add(s, v):
    """Add v to a set s and return s."

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))))))

    assert not empty(s), "Cannot add to an empty set."
    if s.first > v:
        s.first, s.rest = __________________________, __________________________
    elif s.first < v and empty(s.rest):
        s.rest = __________________________
    elif s.first < v:
        __________________________
    return s
Set Operations
**Proposal 2:** A set is represented by a linked list with unique elements that is ordered from least to greatest

```python
def intersect(set1, set2):
    if empty(set1) or empty(set2):
        return Link.empty
    else:
        e1, e2 = set1.first, set2.first
        if e1 == e2:
            return Link(e1, intersect(set1.rest, set2.rest))
        elif e1 < e2:
            return intersect(set1.rest, set2)
        elif e2 < e1:
            return intersect(set1, set2.rest)
```

Order of growth? $\Theta(n)$ (Demo)