Ordered Sets
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 = {'one', 'two', 'three', 'four', 'four'}
>>> s
{'three', 'one', 'four', 'two'}
>>> 'three' in s
True
>>> len(s)
4
>>> s.union({'one', 'five'})
{'three', 'five', 'one', 'four', 'two'}
>>> s.intersection({'six', 'five', 'four', 'three'})
{'three', 'four'}
>>> s
{'three', 'one', 'four', 'two'}
```
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

\( \Theta(1) \)

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

\( \Theta(n) \)

In the worst case: \( v \) does not appear in \( s \)

\text{or}

In the average case: \( v \) 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(s, t):
    if s is Link.empty:
        return Link.empty
    rest = __________________________
    if contains(t, s.first):
        return ________________________
    else:
        return __________intersect(s.rest, t)
        Link(s.first, rest)
    else:
        return rest

Time order of worst-case growth

\[ \Theta(n) \]

The size of the set

\[ \Theta(n^2) \]

If sets are the same size
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

>>> 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>Θ(n)</td>
</tr>
<tr>
<td>adjoin</td>
<td>Θ(n)</td>
</tr>
</tbody>
</table>

(Demo)
Set Operations
Intersecting Ordered Linked Lists

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

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

Order of growth? If s and t are sets of size n, then $\Theta(n)$ (Demo)
Set Mutation
Adding to an Ordered List

\[
\text{add}(s, \emptyset) \quad \text{Try to return the same object as input}
\]
Adding to an Ordered List

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

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

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

`s:`

- first: 0
- rest: 

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

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

```
first: 1 0
rest:  

Link instance

first: 3
rest:  

Link instance

first: 5 4
rest:  

Link instance

first: 1
rest:  

Link instance

first: 5
rest:  

Link instance

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:  |   |
```
Adding to a Set Represented as an Ordered List

def add(s, v):
    """Add v to a set s, returning modified 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))))))

if empty(s): return Link(v)
if s.first > v:
    s.first, s.rest = __________________________ , _____________________________
elif s.first < v and empty(s.rest):
    s.rest = ___________________________________________________________________
elif s.first < v:
    __________________________________________________________________________
    add(s.rest, v)

return s