Lecture #11: Sequences to Trees

Announcements
• Room assignments for Test #1 will be mailed out.
• Next test last week of March (after spring break).
• HKN review session 3-6PM on Saturday in 306.
• Official TA review session 5-7PM tonight (155 Dwinelle).

Review: Sequence Comprehension

• Syntax:
  
  \[
  [ \langle expr \rangle \text{ for } \langle var \rangle \text{ in } \langle sequence \text{ expr} \rangle ] \\
  [ \langle expr \rangle \text{ for } \langle var \rangle \text{ in } \langle sequence \text{ expr} \rangle \text{ if } \langle boolean \text{ expression} \rangle ]
  \]

• Examples:
  
  \[
  [2^x \text{ for } x \text{ in } \text{range}(5)] \Rightarrow [1, 2, 4, 8, 16]
  
  L = [5, 7, 8, 10, 6, 8, 7, 4, 9, 8]
  
  [x \text{ for } x \text{ in } L \text{ if } x \% 2 == 1] \Rightarrow [5, 7, 9]
  
Representing Multi-Dimensional Structures

• How do we represent a two-dimensional table (like a matrix)?
• Answer: use a sequence of sequences (typically a list of lists or tuple of tuples).
• The same approach is used in C, C++, and Java.
• Example:

  \[
  \begin{bmatrix}
  1 & 2 & 0 & 4 \\
  0 & 1 & 3 & -1 \\
  0 & 0 & 1 & 8 \\
  \end{bmatrix}
  \]

  becomes

  \[
  ([1, 2, 0, 4], [0, 1, 3, -1], [0, 0, 1, 8])
  \]

  # or

  \[
  ([1, 2, 0, 4], [0, 1, 3, -1], [0, 0, 1, 8])
  \]

  # or (for old Fortran hands):

  \[
  ([1, 0, 0], [2, 1, 0], [0, 3, 1], [4, -1, 8])
  \]

Problem: Creating A Two-Dimensional Array

```python
def multiplication_table(rows, cols):
    ""
    A ROWS x COLS multiplication table where row x column y
    (element [x][y]) contains xy. Example:
    >>> multiplication_table(4, 3)
    
    [[0, 0, 0], [0, 1, 2], [0, 2, 4], [0, 3, 6]]
    ""
    return
```

Problem: Creating a Triangular Array

• There's no reason the rows in a 2D list must have the same length.

```python
def triangle(rows):
    """A ROWSxROWS lower-triangular array containing """"*""""s."""
    ```

Variation: Creating a Numbered Triangular Array

• This time, use numbers instead of asterisks.

```python
def numbered_triangle(rows):
    """A ROWSxROWS lower-triangular array whose elements
    are integers, starting at 0 going left-to-right,
    up-to-down.
    >>> numbered_triangle(3)
    [[0], [1, 2], [3, 4, 5]]"""
    ```
And Why Stop There? Trees

• We can have rows of rows, and rows of rows of rows, but we needn’t stop at an arbitrary limit.
• Result what is a form of tree:

![Tree Diagram]

• The dots and numbers are generally called vertices or nodes, connected by edges.
• Top node is the root, bottom ones are leaves, others are inner nodes.
• Each node is itself the root of a subtree; those immediately below are its children.

Trees With Labels

• Generally, there can be data at each node, called labels:

![Labelled Tree Diagram]

• How can we represent this structure?

Tree Interface

• Evidently, trees have labels and children, suggesting an API like this:

```python
def make_tree(label, kids = []):
    """A (sub)tree with given LABEL at its root, whose children are KIDS.""
    return [ label ] + kids

def label(tree):
    """The label on TREE.""
    return tree[0]

def children(tree):
    """The immediate descendants of TREE (each a tree).""
    return tree[1:]

def isleaf(tree):
    """True if TREE is a leaf node.""
    return len(tree) == 1
```

• Representation?

Tree Representation

```python
def make_tree(label, kids = []):
    """A (sub)tree with given LABEL at its root, whose children are KIDS.""
    return (label, kids)

def label(tree):
    """The label on TREE.""
    return tree[0]

def children(tree):
    """The immediate descendants of TREE (each a tree).""
    return tree[1]

def isleaf(tree):
    """True if TREE is a leaf node.""
    return len(children(tree)) == 0
```

Alternatives?

Tree Representation (II)

```python
def make_tree(label, kids = []):
    """A (sub)tree with given LABEL at its root, whose children are KIDS.""
    return (label, kids)

def label(tree):
    """The label on TREE.""
    return tree[0]

def children(tree):
    """The immediate descendants of TREE (each a tree).""
    return tree[1]

def isleaf(tree):
    """True if TREE is a leaf node.""
    return len(children(tree)) == 0
```

Algorithms on Trees

• Trees have a recursive structure. A tree is:
  • A label and
  • Zero or more children, each a tree.
• Recursive structure implies recursive algorithm.
Counting Leaves

def count_leaves(tree):
    """The number of leaf nodes in TREE."""