CSC148 fall 2013

linked structures

week 7

Danny Heap
heap@cs.toronto.edu
BA4270 (behind elevators)
http://www.cdf.toronto.edu/~heap/148/F13/
416-978-5899

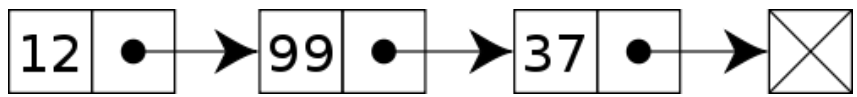
October 22, 2013

Outline

linked lists, conceptually

- **data:** Sequence of nodes, each with a value and reference to next (successor) node. List has reference to front (**aka** head) node).

- **operations:** insert(node), find(value), ...



a node class

```
class LListNode:
    '''Linked List node that can reference next node.'''

    def __init__(self, value=None, nxt=None):
        '''Create a LListNode with value and reference to next LListNode'''

        self.value, self.nxt = value, nxt

    def __repr__(self):
        '''Represent this node as a string.'''

        return 'LListNode(' + str(self.value) + ', ' + str(self.nxt) + ')'
```

prefer `repr` to `str`

Two special methods for representing an object. If you omit `__str__`, Python will use `__repr__` By convention, the latter should be able to produce an equivalent object.

an alternative Stack

Use `LListNode` to re-implement Stack, and compare performance.

histogram for test #1

	q1	q2	q3	t1
out of	10.0	15.0	15.0	45.0
average%	74.9	69.9	69.6	71.5
excl.dr.	74.9	69.9	69.6	71.5
>=100%	84	34	11	22
90..<100%	1	26	11	26
80..<90%	5	50	58	26
70..<80%	13	12	35	36
60..<70%	17	30	43	34
50..<60%	12	5	16	34
40..<50%	35	14	17	15
35..<40%	8	0	3	4
30..<35%	14	7	4	6
20..<30%	16	16	8	3
10..<20%	0	3	0	0
1..<10%	0	3	0	0
< 1%	1	6	0	0

general tree

a linked list is a special case of a tree, with arity (AKA branching factor) of 1. Here's a more general tree node:

```
class TreeNode:
    """Node with children."""

    def __init__(self: 'TreeNode',
                 value: object =None, children: list =None):
        """Node with any number of children"""

        self.value = value
        if not children:
            self.children = []
        else:
            self.children = children[:] # quick-n-dirty copy of list

    def __repr__(self: 'TreeNode'):
        """Represent this TreeNode as a string"""

        return ('TreeNode(' + str(self.value) + ', ' +
                repr(self.children) + ')')
```

nodes need a tree

In order to keep track of the root, and perhaps some methods, we need a Tree class:

```
class Tree:
    """Bare-bones Tree ADT"""

    def __init__(self, root=None):
        """Create a new tree rooted at root."""

        self.root = root
```

try to implement:

```
arity(self: 'Tree') -> int:
    """Return the maximum branching factor of this tree"""
```

```
__contains__(self: 'Tree', value: object) -> bool:
    """Return whether this tree has a node with value"""
```

```
node_count(self: 'Tree') -> int:
    """Number of nodes in this tree"""
```