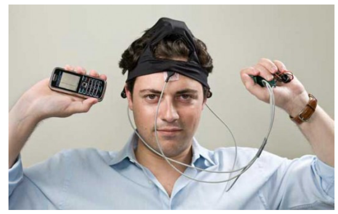
CS61A Lecture 11
Immutable Trees

Jom Magrotker
UC Berkeley EECS
July 5, 2012

Computer Science in the News

iBrain to allow Stephen Hawking to communicate through brainwaves alone

By Kent Sutherland
11:52 July 4, 2012
3 Comments
3 Pictures



http://www.gizmag.com/ibrain-stephen-hawking-communicate-brainwaves/23182/ 2

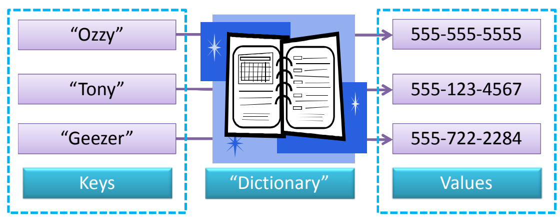
Today

- Review: Immutable Dictionaries
- Deep Tuples
- Immutable Trees

3

Review: Dictionaries

Often we want to associate pieces of data with other pieces of data.



4

Review: Immutable Dictionaries

```
>>> phone_bk = make_idict((“Ozzy”, “555-5555”),
...                       (“Tony”, “123-4567”),
...                       (“Geezer”, “722-2284”))
>>> idict_select(phone_bk, “Ozzy”)
“555-5555”
>>> idict_select(phone_bk, “Geezer”)
“722-2284”
>>> idict_keys(phone_bk)
(“Ozzy”, “Tony”, “Geezer”)
```

5

Review: Higher Order Functions For Sequences

```
>>> nums = (1, 2, 3, 4, 5)
>>> tuple(map(lambda x: x * x, nums))
(1, 4, 9, 16, 25)
>>> tuple(map(lambda x: x + 1, nums))
(2, 3, 4, 5, 6)
>>> tuple(filter(lambda x: x % 2 == 0, nums))
(2, 4)
>>> tuple(filter(lambda x: x <= 3, nums))
(1, 2, 3)
>>> from functools import reduce
>>> reduce(lambda x, y: x * y, nums, 1)
120
>>> reduce(lambda x, y: x + y, nums, 0)
15
```

6

Practice: Higher Order Functions For Sequences

What are the outputs for each of the following lines of Python?

```
>>> from operator import add
>>> tuple(map(lambda x: reduce(add, x),
        ((2, 3), (5, 6), (8, 9))))
?????
>>> tuple(map(lambda x: x - 1,
              filter(lambda x: x % 2 == 0,
                     map(lambda x: x + 1,
                         range(10)))))
?????
```

7

Practice: Higher Order Functions For Sequences

What are the outputs for each of the following lines of Python?

```
>>> from operator import add
>>> tuple(map(lambda x: reduce(add, x),
        ((2, 3), (5, 6), (8, 9))))
(5, 11, 17)
>>> tuple(map(lambda x: x - 1,
              filter(lambda x: x % 2 == 0,
                     map(lambda x: x + 1,
                         range(10)))))
(1, 3, 5, 7, 9)
```

8

Announcements

- Homework 5 is due **July 6**.
- Project 2 is due **July 13**.
- Project 1 contest is on!
  - *How to submit*: Submit a file pig.py with your `final_strategy` to `proj1-contest`.
  - *Deadline*: Friday, **July 6** at **11:59pm**.
  - *Prize*: One of 3 copies of *Feynman* and 1 extra credit point.
  - *Metric*: We will simulate your strategy against everyone else's, and tally your win rate. Draws count as losses.

9

Announcements: Midterm 1

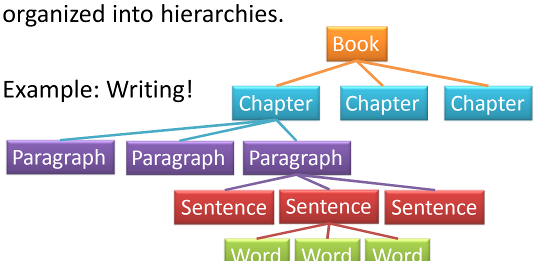
- Midterm 1 is on **July 9**.
  - *Where*? 2050 VLSB.
  - *When*? 7PM to 9PM.
  - *How much*? Material covered until July 4.
- Closed book and closed electronic devices.
- One 8.5" x 11" 'cheat sheet' allowed.
- Group portion is 15 minutes long.
- Post-midterm potluck on Wednesday, **July 11**.

10

Hierarchical Data

Often we find that information is nicely organized into hierarchies.

Example: Writing!

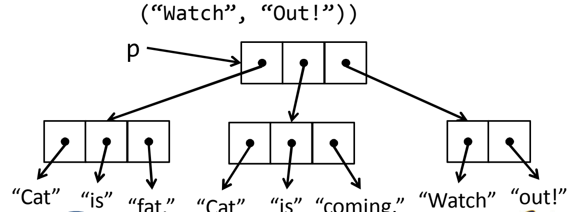


11

Hierarchical Data: Deep Tuples

We already had a way of representing something like this.

```
>>> p = (("Cat", "is", "fat."),
         ("Cat", "is", "coming."),
         ("Watch", "Out!"))
```



12

Operating on Deep Tuples

So we already have a simple way of organizing data into hierarchies using "deep tuples."

How do we manipulate deep tuples?

- Not that different from working with regular tuples.
- Use ***tree recursion***!

Example: Operating on Deep Tuples

Let's say I want to write a function, `evens_count`, that counts the number of even numbers found in a deep tuple containing numbers (or tuples).

```
>>> woah_deep = ((1, 2), 3, ((4, 5), 6))
>>> evens_count(woah_deep)
3
```


Example: Operating on Deep Tuples

How we would have solved this if we were handed a simple tuple of numbers?

Now extend this to handle deep tuples!

```
# Iterative
def evens_count(t):
    total = 0
    for num in t:
        if num % 2 == 0:
            total += 1
    return total
```

```
# Recursive
def evens_count(t):
    if len(t) == 0:
        return 0
    if t[0] % 2 == 0:
        return 1 + evens_count(t[1:])
    return 0 + evens_count(t[1:])
```


Example: Operating on Deep Tuples

```
# Recursive
def evens_count(t):
    if len(t) == 0:
        return 0
    if is_tuple(t[0]):
        return evens_count(t[0]) + evens_count(t[1:])
    if t[0] % 2 == 0:
        return 1 + evens_count(t[1:])
    return 0 + evens_count(t[1:])
```

```
def is_tuple(x):
    return type(x) is tuple
```

First check if the first item in the sequence is also a tuple. If so, use ***tree recursion*** and count the evens in both the first item and the rest of t.

Practice: Operating on Deep Tuples

Write the procedure `deep_filter`, which takes a predicate and a deep tuple and returns a new deep tuple with only the items for which predicate returns True.

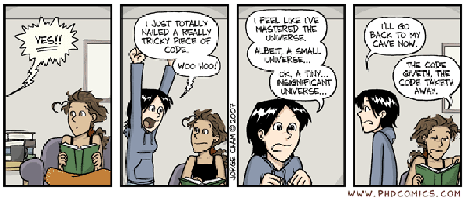
```
>>> woah_deep = ((1, 2), 3, ((4, 5), 6))
>>> deep_filter(lambda x: x % 2 == 0,
                woah_deep)
((2,), ((4,), 6))
>>> deep_filter(lambda x: x >= 2 and x <= 3,
                woah_deep)
((2,), 3, ((),))
```


Practice: Operating on Deep Tuples

Write the procedure `deep_filter`, which takes a predicate and a deep tuple and returns a new deep tuple with only the items for which predicate returns True.

```
def deep_filter(pred, dt):
    if len(dt) == 0:
        return dt
    if is_tuple(dt[0]):
        return (deep_filter(pred, dt[0]),) +\
               deep_filter(pred, dt[1:])
    if pred(dt[0]):
        return (dt[0],) + deep_filter(pred, dt[1:])
    return deep_filter(pred, dt[1:])
```

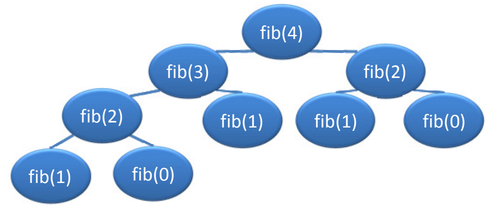
Break



19

Hierarchical Data: Trees

Often, deep tuples aren't quite expressive enough for our purposes. Sometimes we want values in the middle too!

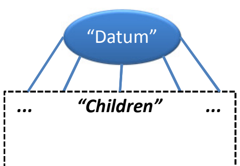


20

Hierarchical Data: Trees

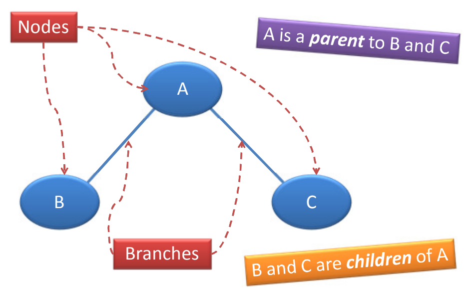
A ***tree*** data structure traditionally has 2 parts:

1. A ***datum*** – The data stored in the top point of the tree.
2. Some ***children*** – Any trees that appear below this tree.



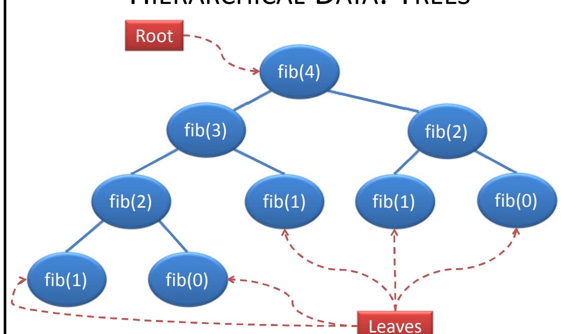
21

Hierarchical Data: Trees



22

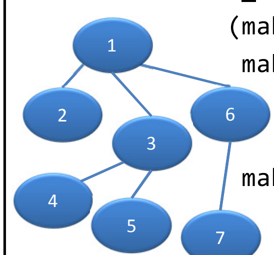
Hierarchical Data: Trees



23

ITrees

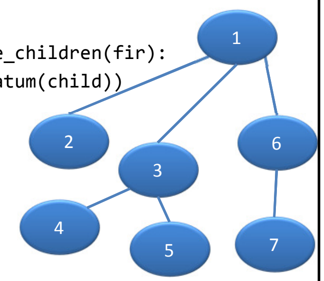
```
>>> fir = make_tree(1,
              (make_tree(2),
               make_tree(3,
                   (make_tree(4),
                    make_tree(5))),
               make_tree(6,
                   (make_tree(7),))))
```



24

ITREES

```
>>> itree_datum(fir)
1
>>> for child in itree_children(fir):
        print(itree_datum(child))
2
3
6
```



ITREES

```
def make_itree(datum, children=()):
    return (datum, children)

def itree_datum(t):
    return t[0]

def itree_children(t):
    return t[1]
```

Constructor

Selectors

EXAMPLE: OPERATING ON ITREES

Suppose I want to write the function `itree_prod`, which takes an ITree of numbers and returns the product of all the numbers in the ITree.

```
>>> t = make_itree(1, (make_itree(2),
                       make_itree(3),
                       make_itree(4)))
>>> itree_prod(t)
24
```

Look! I got it right this time!

EXAMPLE: OPERATING ON ITREES

Idea: split the problem into 2 different parts: handling a single tree and handling a group of trees (a *forest*).

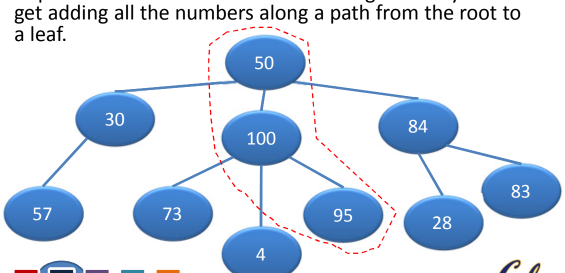
```
def itree_prod(t):
    return itree_datum(t) * forest_prod(itree_children(t))

def forest_prod(f):
    if len(f) == 0:
        return 1
    return itree_prod(f[0]) * forest_prod(f[1:])
```

This is called ***mutual recursion*** because it involves two functions recursively calling each other!

PRACTICE: OPERATING ON ITREES

Write the function `max_path_sum`, which takes an ITree of positive numbers and returns the largest sum you can get adding all the numbers along a path from the root to a leaf.



PRACTICE: OPERATING ON ITREES

Write the function `max_path_sum`, which takes an ITree of positive numbers and returns the largest sum you can get adding all the numbers along a path from the root to a leaf.

```
def max_path_sum(t):
    ??????


def max_forest_sum(f):
    if len(f) == 0:
        return 0
    ??????
```

Practice: Operating on ITrees

Write the function `max_path_sum`, which takes an ITree of positive numbers and returns the largest sum you can get adding all the numbers along a path from the root to a leaf.

```
def max_path_sum(t):
    max_child_sum = max_forest_sum(itree_children(t))
    return itree_datum(t) + max_child_sum

def max_forest_sum(f):
    if len(f) == 0:
        return 0
    return max(max_path_sum(f[0]),
               max_forest_sum(f[1:]))
```

31

Conclusion

- Organizing data into hierarchies is ***very*** useful and ***very*** common in Computer Science
- We can think of nested tuples as a simple form of a tree structure that only has leaves.
- ITrees are useful for representing general tree-structures.
- ***Preview:*** binary search trees!

32