Efficiency

Announcements

Tree Class

Tree Class

A Tree has a label and a list of branches; each branch is a Tree

Tree class

```python
class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)



def fib_tree(n):
    if n == 0 or n == 1:
        return Tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = left.label + right.label
        return Tree(fib_n, [left, right])
```

tree data abstraction

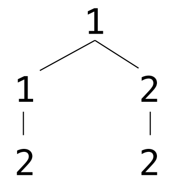
```python
def tree(label, branches=[]):
    for branch in branches:
        assert is_tree(branch)
    return [label] + list(branches)
def label(tree):
    return tree[0]
def branches(tree):
    return tree[1:]
def fib_tree(n):
    if n == 0 or n == 1:
        return tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = label(left) + label(right)
        return tree(fib_n, [left, right])
```

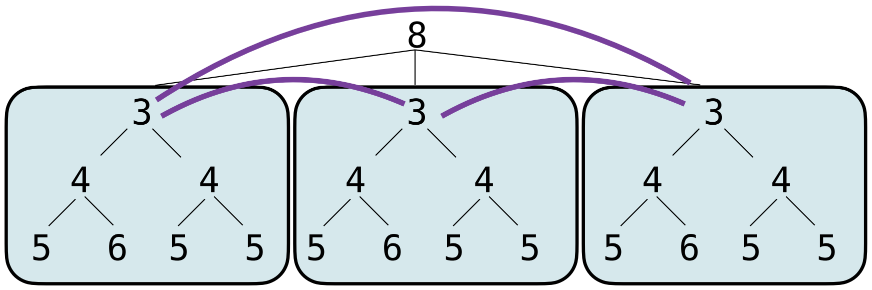
Example: Count Twins

Implement twins, which takes a Tree t. It return the number of pairs of sibling nodes whose labels are equal. **Definition:** Two nodes are siblings if they have the same parent.

```
def twins(t):
    """Count the pairs of sibling nodes with equal labels.

    >>> t1 = Tree(3, [Tree(4, [Tree(5), Tree(6)]), Tree(4, [Tree(5), Tree(5)])])
    >>> twins(t1)  # 4 and 5
    2
    >>> twins(Tree(1, [Tree(1, [Tree(2)]), Tree(2, [Tree(2)])]))
    0
    >>> twins(Tree(8, [t1, t1, t1]))  # 3 pairs of twins at the top, plus 2 in each branch
    9
    """
    count = 0
    n = len(t.branches)
    for i in range(n-1):
        for j in range(i+1, n):
            if t.branches[i].label == t.branches[j].label:
                count += 1
    return count + sum([twins(b) for b in t.branches])
```





pollev.com/cs61a

Linked List Example: Cycles

Create a List and Linked List

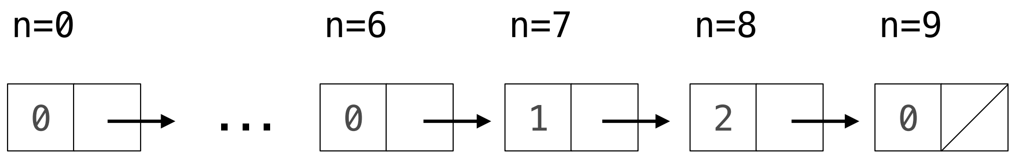
```
def cycle(k, n):
    """Build an n-element list that cycles among range(k).

    >>> cycle(3, 10)
    [0, 1, 2, 0, 1, 2, 0, 1, 2, 0]
    """
    s = []
    for i in range(n):
        s.append(i % k)  # Add to the end
    return s
```

```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        assert ... # rest is a linked list
        self.first = first
        self.rest = rest
```

```
def cycle_link(k, n):
    """Build an n-element linked list that cycles among range(k).

    >>> print(cycle_link(3, 10))
    (0 1 2 0 1 2 0 1 2 0)
    """
    s = Link.empty
    while n > 0:
        n -= 1
        s = Link(n % k, s)  # Add to the front
    return s
```



Create a Linked List by Appending

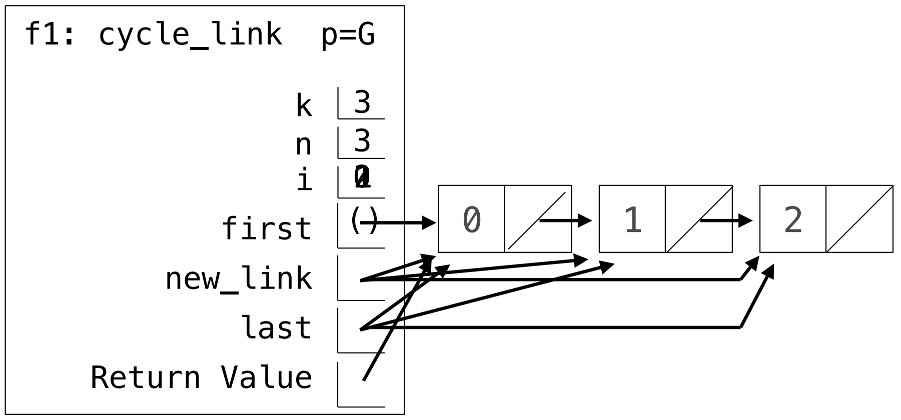
```
def cycle(k, n):
    """Build an n-element list that cycles among range(k).

    >>> cycle(3, 10)
    [0, 1, 2, 0, 1, 2, 0, 1, 2, 0]
    """
    s = []
    for i in range(n):
        s.append(i % k)  # Add to the end
    return s
```

```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        assert ... # rest is a linked list
        self.first = first
        self.rest = rest
```

```
def cycle_link(k, n):
    """Build an n-element linked list that cycles among range(k).

    >>> print(cycle_link(3, 10))
    (0 1 2 0 1 2 0 1 2 0)
    """
    first = Link.empty
    for i in range(n):
        new_link = Link(i % k)
        if first is Link.empty:
            first, last = new_link, new_link
        else:
            last.rest = new_link  # Add to the end
            last = new_link
    return first
```

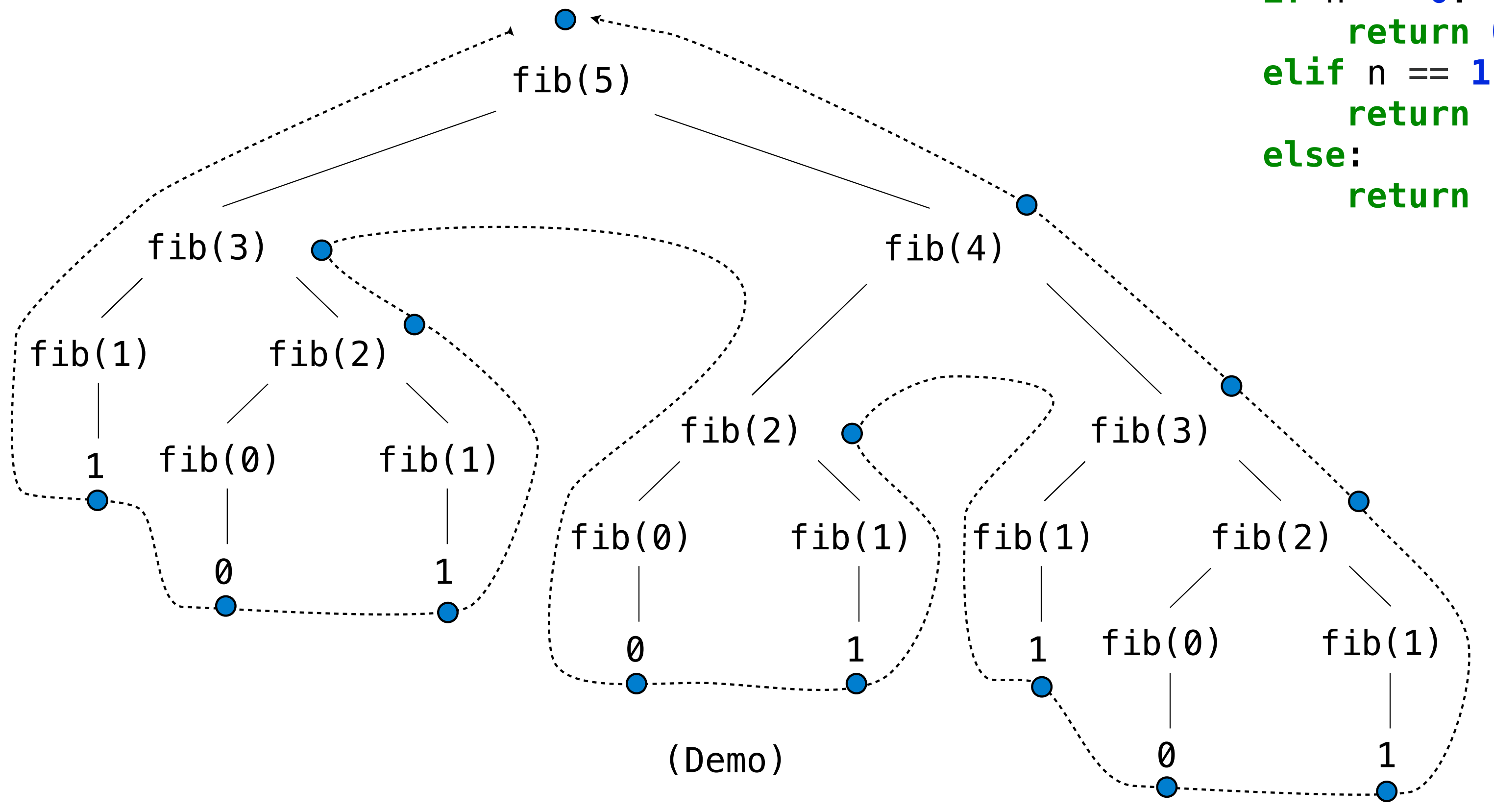


Measuring Efficiency

Recursive Computation of the Fibonacci Sequence

Our first example of tree recursion:

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```



(Demo)

Memoization

Memoization

Idea: Remember the results that have been computed before

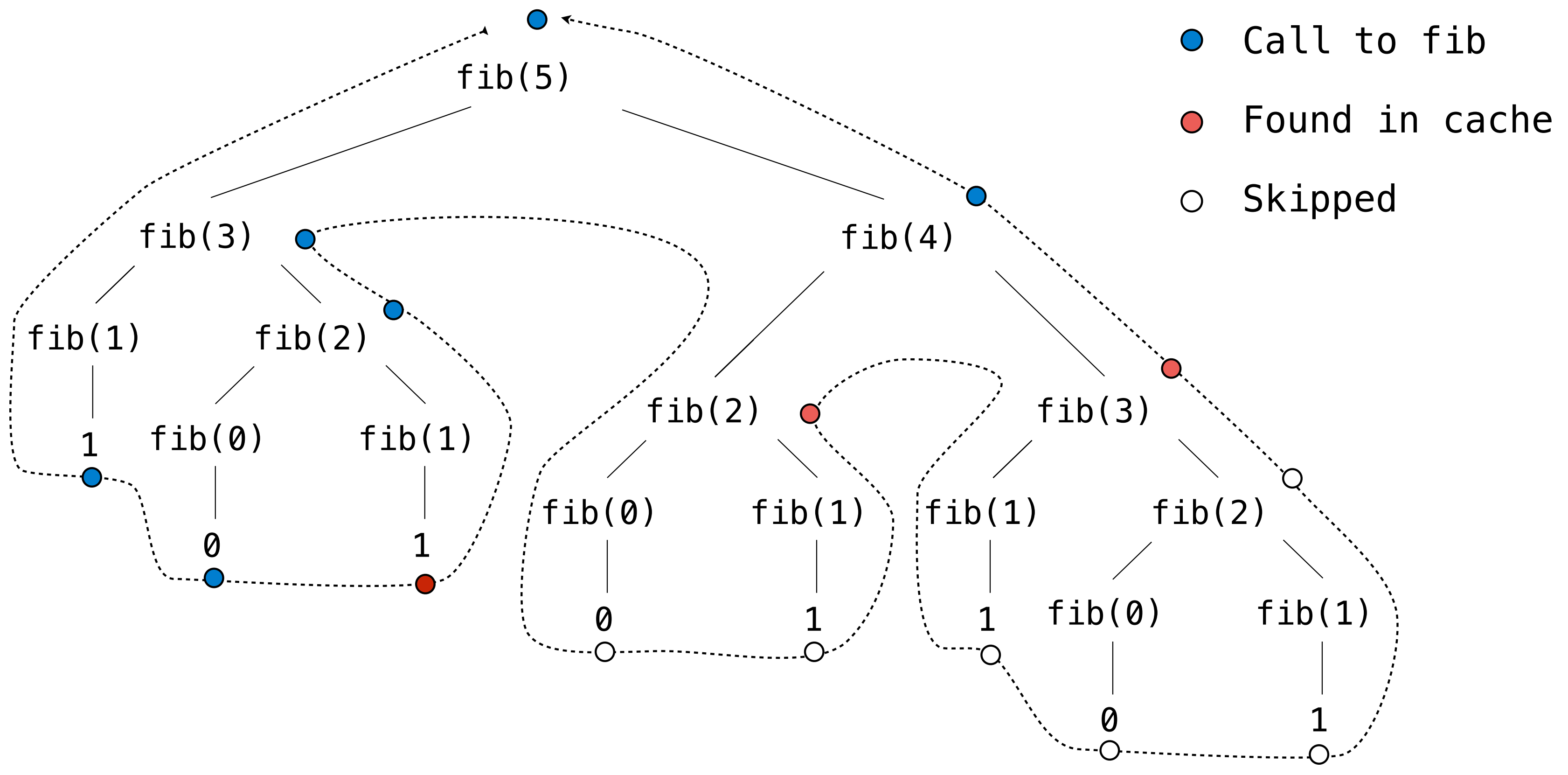
```
def memo(f):
    cache = {}
    def memoized(n):
        if n not in cache:
            cache[n] = f(n)
        return cache[n]
    return memoized
```

Keys are arguments that map to return values

Same behavior as f, if f is a pure function

(Demo)

Memoized Tree Recursion



Orders of Growth

Common Orders of Growth

Common examples:

Exponential growth. E.g., recursive fib

Incrementing *n* multiplies *time* by a constant

Tree recursion (without memoization)

Quadratic growth.

Incrementing *n* increases *time* by *n* times a constant

Two nested loops:
For each element in a list,
  process a whole list

Linear growth.

Incrementing *n* increases *time* by a constant

For each element in a list,
  do a fixed amount of work

Logarithmic growth.

Doubling *n* only increments *time* by a constant

Each step shrinks the problem
  by half (or some fraction)

Constant growth. Increasing *n* doesn't affect time

Just process the first few
  elements of a list

Repeated Inserts (Revisited)

```
double(      cycle(      5, 100000), 3): 302ms
double_link(cycle_link(5, 100000), 3):  15ms
```

Double a List and a Linked List (Last Lecture)

```
def double(s, v):
    """Insert another v after each v.

    >>> s = [2, 7, 1, 8, 2, 8]
    >>> double(s, 8)
    >>> s
    [2, 7, 1, 8, 8, 2, 8, 8]
    """
    i = 0
    while i < len(s):
        if s[i] == v:
            s.insert(i+1, v)
            i += 2
        else:
            i += 1
```

Quadratic Growth

Shift over everything after i+1

```
def double_link(s, v):
    """Insert another v after each v.

    >>> end = Link(1, Link(8, Link(2, Link(8))))
    >>> t = Link(2, Link(7,end))
    >>> double_link(t, 8)
    >>> print(t)
    (2 7 1 8 8 2 8 8)
    """
    while s is not Link.empty:
        if s.first == v:
            s.rest = Link(v, s.rest)
            s = s.rest.rest
        else:
            s = s.rest
```

Linear Growth

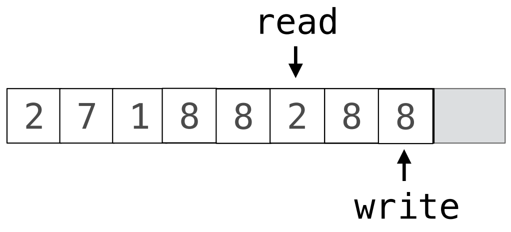
Make 1 Link

Double a List with Linear Growth

Could you insert another v after each v in a Python list s with linear growth?

```
def double_fast(s, v):
    """Insert another v after each v in s.

    >>> s = [2, 7, 1, 8, 2, 8]
    >>> double_fast(s, 8)
    >>> s
    [2, 7, 1, 8, 8, 2, 8, 8]
    """
    read = len(s) - 1
    vs = s.count(v)
    s.extend([0 for _ in range(vs)]) # Make space
    write = len(s) - 1
    while write > read:
        if s[read] == v:
            s[write] = v
            s[write - 1] = v
            write -= 2
        else:
            s[write] = s[read]
            write -= 1
        read -= 1
```



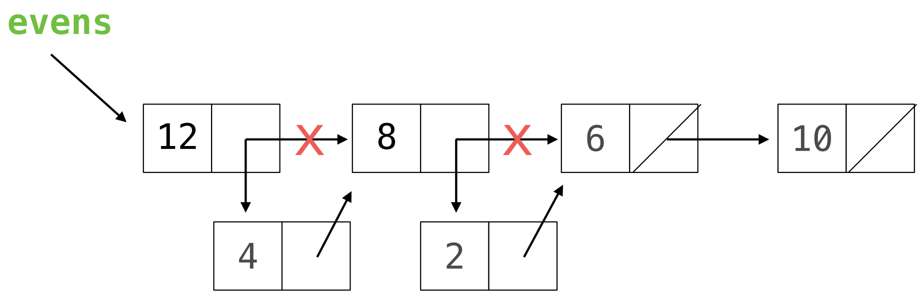
```
double(     cycle(     5, 100000), 3): 302ms
double_link(cycle_link(5, 100000), 3):  15ms
double_fast(cycle(     5, 100000), 3):   8ms
```

Linked List Practice

Inserting into a Linked List

```
def insert_link(s, x, i):
    """Insert x into linked list s at index i.

    >>> evens = Link(4, Link(2, Link(6)))
    >>> insert_link(evens, 8, 1)
    >>> insert_link(evens, 10, 4)
    >>> insert_link(evens, 12, 0)
    >>> insert_link(evens, 14, 10)
    Index out of range
    >>> print(evens)
    (12 4 8 2 6 10)
    """
    if s is Link.empty:
        print('Index out of range')
    elif i == 0:
        second = Link(s.first, s.rest)
        s.first = x
        s.rest = second
    elif i == 1 and s.rest is Link.empty:
        s.rest = Link(x)
    else:
        insert_link(s.rest, x, i-1)
```

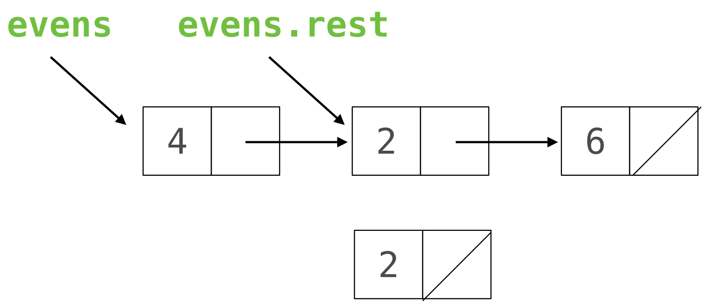


Slicing a Linked List

Normal slice notation (such as s[1:3]) doesn't work if s is a linked list.

```
def slice_link(s, i, j):
    """Return a linked list containing elements from i:j.

    >>> evens = Link(4, Link(2, Link(6)))
    >>> slice_link(evens, 1, 100)
    Link(2, Link(6))
    >>> slice_link(evens, 1, 2)
    Link(2)
    >>> slice_link(evens, 0, 2)
    Link(4, Link(2))
    >>> slice_link(evens, 1, 1) is Link.empty
    True
    """
    assert i >= 0 and j >= 0
    if _j == 0_ or s is Link.empty:
        return Link.empty
    elif _i == 0_:
        return Link(s.first, _slice_link(s.rest, i, j-1)_)
    else:
        return slice_link(s.rest, i-1 , j-1 )
```



```
slice_link(evens, 1, 2) returns
slice_link(evens.rest, 0, 1) links 2 to
slice_link(evens.rest.rest, 0, 0) returns Link.empty
```

pollev.com/cs61a