Lecture 16: Linked Lists

July 19, 2022

Richard Roggenkemper

Announcements

- Midterm grades are released
  - Regrades are due Friday at 11:59pm
- CATS is due tonight!
- Lab 6 is also due tonight!
- Homework 4 is due Thursday night!
- Extended Instructor Office Hours today from 1-4 PM in Soda 784/781

Mid-Semester Feedback

General Feedback

- Lecture
  - Slides are always posted on the Piazza thread before the website
  - Recordings will always be accessible in the Bcourses Media Gallery
  - Lecture will not be webcast (see @28 on Piazza)
- Assignments
  - Labs are taking about 1-1.5 hours
  - Homeworks about 2-4 hours
  - Projects 6-10 hours
  - Unfortunately, posting the assignments really early is not possible

Feedback

- TA + Tutor Feedback has been sent to course staff so they can continue doing the good things and try to improve
- Course Climate
- Reminders
  - Getting Started Videos exist for HW + Projects
    - Will try to get those out earlier
  - Lots of different OH Options
  - Tutoring Sections still have spots open

Richard's Feedback

- Some people mentioned wanting more coding problem examples in lecture (in addition to demos and concepts)
- Will post a code file with each lecture
- Brief review of previous lecture at the beginning of lecture
- I'll try to time the break closer to the middle of lecture
- Slides are always going to be on the Piazza thread before the website updates
- End of lecture rush
- More instructor office hours at different times during the week

Midterm

Thoughts?

My Opinion on the Midterm

- Y’all did really well for a tough exam
- About where we wanted students to be
- Very possible that you did better or worse than you wanted on the test
- We still have just under 4 weeks until the final and a lot can change
    - You might’ve only started learning how to code 4 weeks ago!

My Thoughts

- 61A (and CS exams in general) are pretty different (read “difficult”) at Berkeley than you might be used to
- Best way to practice is looking at past exams
- Averages might be lower than you are used to, but this is normal
- The final will be cumulative and there is a clobber
- Keeping coming to class
- Advantageous to study the topics you didn’t do as well as you wanted to on the midterm

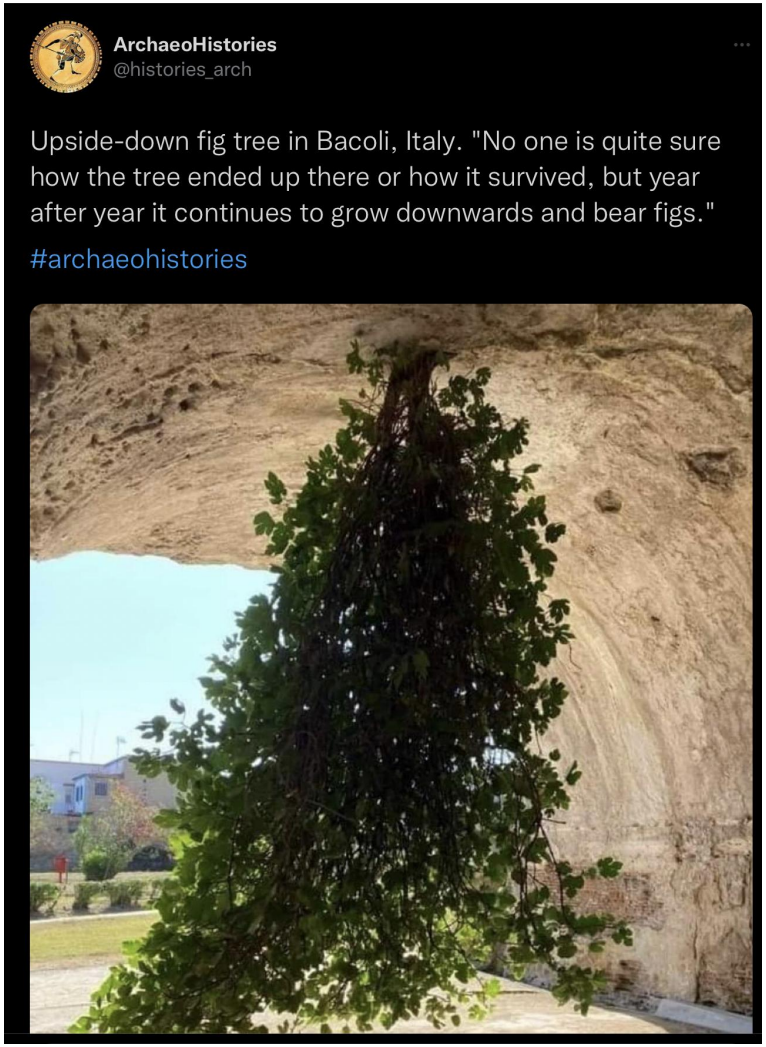
Quick Review

Trees

- Recursive data structure built with Object Oriented Programming
  - Attributes
    - Label - Value
    - Branches - List of Children Trees
- 2 big types of questions
  - 1) Creating a new tree / mutating a tree
  - 2) Solving some problem about the tree
- Both types of problems involve looking at all the branches in a recursive way

Review Problem

- Given a Tree, t, write a function that squares the value of each node on the Tree
- Do this by creating a new Tree and by mutating the original Tree



Linked Lists

What was wrong with lists?

“Don’t”	“Talk”	“About”	“Bruno”
0	1	2	3
1800	1801	1802	1803

Imagine we have this list that has the values of the list, the index, and where in the computer the list is being stored

What happens when we want to add “We” to the front of the list?

“We”	“Don’t”	“Talk”	“About”	“Bruno”
0	1	2	3	4
1800	1801	1802	1803	1804

Why is this bad?

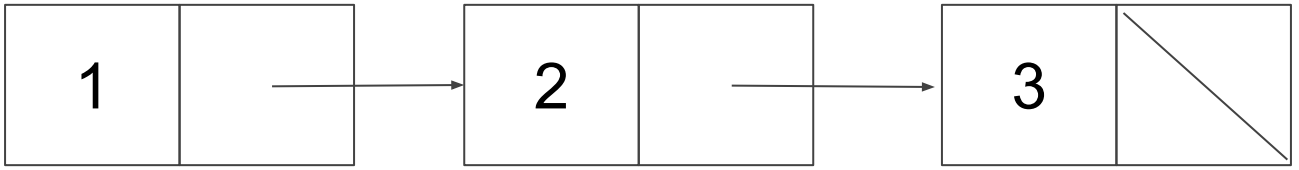
- Inserting items is really slow, especially when it is close to the front of the list

How can we fix this?

- We can store the value anywhere on our computer and just tell it where the “next” element is

Linked Lists

A **linked list** is either empty or a **first** value and the **rest** of the Linked List



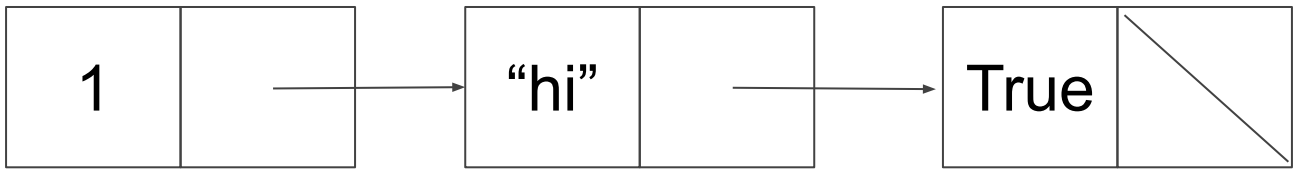
The other values of my Linked List aren't necessarily stored right next to each other, but it still knows where to look

What is **first**?

first in linked lists is where you store the value of the node

Similar to how you would store the label of a tree or a value at an index of a list

first can be a number, a boolean, a function, another linked list, or anything else!

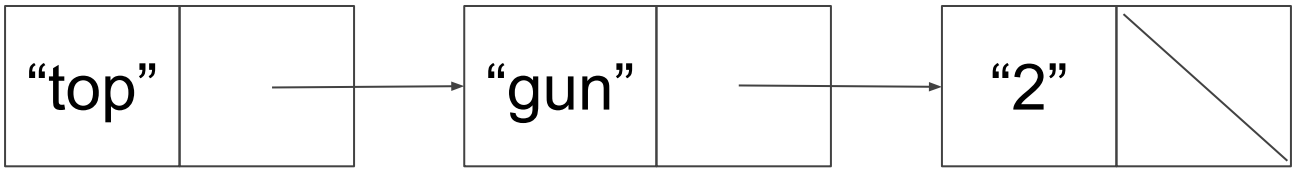


What is **rest**?

rest is where we store the remainder of the linked list

Similar to how the branches of a tree have to also be a tree, the **rest** of a linked list must also be a linked list

Remember that a linked list can also be empty, so when you want to end the linked list, you can set the **rest** to be empty



Demo

Let’s put it all together

```
class Link:
    empty = ()

    def __init__(self, first, rest= empty):
        self.first = first
        self.rest = rest
```

Making it better

```
class Link:
    empty = ()

    def __init__(self, first, rest= empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest
```

Break

String Representation

```
def __repr__(self):
    if self.rest:
        rest_repr = ", " + repr(self.rest)
    else:
        rest_repr = ""
    return "Link(" + repr(self.first) + rest_repr + ")"
```

Demo

String Representation

```
def __str__(self):
    string = "<"
    while self.rest is not Link.empty:
        string += str(self.first) + " "
        self = self.rest
    return string + str(self.first) + ">"
```

Wait… so are they recursive?

How to Solve Linked List Problems

Luckily, we have 2 way to solve linked list problems: iteration and recursion

Some problems might have a better option than the other, but for a lot of problems we can either use iteration or recursion

The difference between Trees and linked lists is that there is only one “branch”

This means that I don’t really need the loop and recursive calls to search the entire linked list, there is only 1 place to go

In this class, you will likely see the recursive solution more often

Demo

Linked List Mutation

Similar to Trees, because first and rest are attributes, we can change them and mutate our linked lists

Some problems will ask for you to mutate the linked list while others will ask you to create a new one

Linked List Range

```
def range_link(start, end):
    """Return a Link containing consecutive integers
    from START to END, not including END.
    >>> range_link(3, 6)
    Link(3, Link(4, Link(5)))
    """
```

Linked List Map

```
def map_link(f, s):
    """Return a Link that contains f(x) for each x in Link s.
    >>> square  = lambda x: x * x
    >>> map_link(square, range_link(3, 6))
    Link(9, Link(16, Link(25)))
    """
```

Linked List Filter

```
def filter_link(f, s):
    """Return a Link that contains only the elements x of
    link s which f(x) is a true value.

    >>> id_odd  = lambda x: x % 2 == 1

    >>> filter_link(is_odd, range_link(3, 6))

    Link(3, Link(5))

    """
```

Demo

Summary

- Linked Lists give us a recursive object that stores values like a list, but with some speed benefits
- Can be solved iteratively or recursively, but you will see more recursion in this class
- Mutable objects so we can directly change values instead of creating a new linked list