Discussion 9

Linked Lists, Efficiency, and Mutable Trees

Antonio Kam

anto [at] berkeley [dot] edu

All slides can be found on

`teaching.rouxl.es`

Announcements

- Ants due on Friday, EC point for submitting on Thursday
- HW will cover linked lists/mutable trees, you'll get more practice on this

Notes from last section

- Cereal or Milk first
  - cereal first easily
  - i think cereal can be nice even without milk, so milk is just an add-on for later for me
- (help i dont get mutable trees, it's kinda hard)
  - makes sense - they are hard and definitely a weird concept to get used to (especially because you've mainly dealt with the data abstraction version)
  - the thing here is instead of returning a value, you could be mutating a value instead
  - this is perfectly fine - can still treat the process the same way!
  - do the hw - it'll give you more practice; email me if you want

Notes from last section

- more solo time
  - discussion isn't the place for more solo time typically
  - ask away during lab! i'm more than happy to talk about anything during lab (and the other AIs are too!)
- favorite song/soundtrack
  - Ori and the Blind Forest/Ori and the Will of the Wisps
  - it's a game soundtrack (orchestral) that i've already listened to many times (and still listen to a lot)

Temperature Check 🌡️

- Linked lists (Covered in today's lecture)
- Efficiency
- Mutable Trees

Linked Lists

Linked Lists

- Linked lists are recursive data structures
- You can use linked lists to create your own version of a sequence
- They are generally useful when you want to have an infinitely-sized list, or want to dynamically change the size of the list (more useful in 61B if you do end up taking it)
- In general, linked lists problems can be solved using both iteration and recursion
  - Like trees, they are recursive data structures, but unlike trees, you can use both recursion and iteration to solve them.

Linked Lists (Anatomy)

```
class Link:
    empty = ()

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

- Linked lists have a `first` (similar to `label`) attribute and a `rest` (similar-ish to `branches`) attribute.

Linked Lists (Construction)

Note to Anto: Draw box-and-pointer diagram

```
s = Link(1, Link(2, Link(3)))
>>> s.first
1
>>> s.rest
Link(2, Link(3))
>>> s.rest.first
2
s2 = Link(1, 2) # This will error because rest is not a linked list
>>> s.rest.rest.first
3
```

Worksheet!

Results from last section (`links.rouxl.es/disc`)

- Baked (2)
- Scalloped (2)
- "Violently diced via high speed pressurized potato cannon at the McDonalds fry factory, fried at 400 degrees Fahrenheit, then flash frozen."
- Mashed
- Potato Chips/French Fries/Tater Tots
- Well Done
- Microwaved
- Boiled
- "Do NOT boil me!!! Anything else is ok"

Efficiency

Efficiency

- In general, your programs take time to run, but some perform better than others.
  - While at a small scale, you can't really tell the difference in terms of time, when your inputs become really big, the 'runtime factor' of your algorithm starts to really matter in the amount of time it takes
- There are different types of runtime, but the ones we'll be focusing on are:
  - Constant
  - Logarithmic
  - Linear
  - Quadratic
  - Exponential

Efficiency - Constant

```python
def constant(n):
    for i in range(50):
        print(50)
```

- Notice how no matter what you do to the input of `n`, you'll only be running through the for loop `50` times. This is therefore **constant** in runtime.

Efficiency - Logarithmic

```
def log(n):
    while n > 0:
        print(n % 10)
        n //= 10
```

- For this one, when we go through the while loop, we're dividing by 10 each time.

Efficiency - Linear

```
def linear(n):
    while n > 0:
        print(n)
        n -= 1
```

- In this case, the input size scales linearly to the input that you give

Efficiency - Quadratic

```
def quadratic(n):
    for i in range(n):
        for j in range(n):
            print(i + j)
```

- In this case, the amount of steps needed scales quadratically

Efficiency - Exponential

```
def exponential(n):
    assert n >= 0 and type(n) == int
    if n == 0 or n == 1:
        return n
    else:
        return exponential(n - 1) + exponential(n - 2)
```

- As the input size gets larger, we have to make significantly more calls
- This is very inefficient and should be avoided where possible (sometimes it is not possible to avoid exponential time)

Worksheet!

Trees

Trees (construction)

- The `Tree` constructor takes in a `label`, and an *optional* `list` of `branches`. If `branches` isn't given, it defaults to an empty list `[]`

```
class Tree: # simplified version compared to the 61A implementation
    def __init__(self, label, branches = []):
        assert type(branches) == list
        for item in branches:
            assert isinstance(item, Tree)
        self.label = label
        self.branches = branches
```

Construction: `Tree(2)`; `Tree(2, [Tree(3), Tree(4)])`

Note that the branches **must** be in a list.

Trees (access)

```
t = Tree(3, [Tree(4, [Tree(5)]), Tree(6)])
>>> t.label
3
>>> t.branches
[Tree(4, [Tree(5)]), Tree(6)]
```

Worksheet!

Mental Health Resources

- CAPS:
  - If you need to talk to a professional, please call CAPS at 510-642-9494.
- After Hours Assistance
  - For any assistance after hours, details on what to do can be found at this link

Anonymous Feedback Form

`links.rouxl.es/feedback`

Thanks for coming! 🥳

Please give me feedback on what to improve!