CS 61A
Spring 2021

Lists and Trees

Lost 4: February 26, 2021

1 Learning Goals

- Discuss strategies for improving exam scores
- Introduce lists and trees
- Understand what it means for an object to be *mutable*.
- Understand how to use list comprehensions effectively
- Understand how to approach difficult tree questions

2 Midterm Improvement Tips

If the midterm did not go as well as you hoped, **do not stress**. There is still plenty of time to improve your test-taking skills. Oftentimes, students struggle with Midterm 1 because they feel they knew all the material but were just thrown off by the structure and time limitations of the exam. Below are some study tips that you may find useful for future exams.

- Continue attending class and learning the material as you have been doing.
- Set aside an hour of time each week to attempt 1-2 exam-level problems on the week's topic *on your own*.
- When you study for Midterm 2 via practice exams, do the following:
  - **Before taking each exam, simulate the actual exam setting as best as you can**. Lock your door, restrict yourself to only being able to access 61A's website for reference, type out your answers on a word doc (to simulate the 61A Exam Tool) instead of on paper, and so on. This will mentally make you familiar with the setting and reduce tension and nerves when you actually take the exam.
  - **Time yourself**. The importance of this cannot be emphasized enough. You have to time yourself, and when you run out of time, self-grade your exam **before** attempting the rest of the problems, so you can accurately track how your grade improves as you take more practice exams. Some people time themselves by question, but this is not good enough. By timing the entire exam, you force yourself to make decisions and practice techniques you will need to make/use on the actual exam: Should I skip this question and come back to it? How quickly can I check the doctests? If I have a total of 25 minutes and 3 questions, which ones should I prioritize?
  - **Re-attempt questions 2-3 days later**. Mark all of the questions you get incorrect on your practice exams. For each one, carefully read and understand the solution, and then give yourself 2-3 days to forget it. Then attempt the problem again without looking at the solution. This ensures that you actually internalize the approach.
  - **Do not get stuck on any one question**. If you've spent 2-3 minutes reading and re-reading a problem and still do not know how to approach it, it is time to skip it and move on. This is an important skill which maximizes your exam score. You want to make sure you have enough time to attempt all the problems you do know how to do. Come back to the others later.

This seems like a lot, but it is not terrible if you plan ahead and spread it out over time (you should never take more than one practice exam in a day). We encourage you to keep giving your best effort to the course; with practice, it does get easier. Furthermore, exams are designed to be difficult for various reasons, but a low score on them should not discourage you. The skills you learn in this class are far more valuable than the examscores, so keep that in mind. Best of luck!

3 Lists and Trees Practice

If we'd like to get back parts of the list, as opposed to single elements, we *slice* the list. Slicing a list returns us a **copy** of a chunk of the original list. We use the following syntax:

```
lst[start:end:step]
```

where start, end, and step are integers. The slice includes elements at indices start, start+1*step, start+2*step, and so on up to end. It is legal to omit one or more of start, end, and step; they default to 0, `len(lst)`, and 1, respectively. Start and end can be negative, meaning you count from the end.

```
>>> a = [0, 1, 2, 3, 4, 5, 6]
>>> a[1:4]
[1, 2, 3]
>>> a[1:6:2]
[1, 3, 5]
>>> a[:4]
[0, 1, 2, 3]
>>> a[3:]
[3, 4, 5, 6]
>>> a[-1:]
[6]
>>> a
[0, 1, 2, 3, 4, 5, 6]
```

3.1 Name two data types that are mutable. What does it mean to be mutable?

3.2 Name at least two data types at are not mutable.

3.3 Will the following code error? If so, why?

```
>>> a = 1
>>> b = 2
>>> dt = {a: 1, b: 2}
```

```
>>> a = [1]
>>> b = [2]
>>> dt = {a: 1, b: 2}
```

3.4 Fill in the output and draw a box-and-pointer diagram for the following code. If an error occurs, write "Error", but include all output displayed before the error.

```
a = [1, [2, 3], 4]
c = a[1]
```

```
c

a.append(c)
a

c[0] = 0
c

a

a.extend(c)
c[1] = 9
a

list1 = [1, 2, 3]
list2 = [1, 2, 3]
list1 == list2

list1 is list2
```

3.5 Check your understanding:

1 What is the difference between the append function, extend function, and the '+' operator?

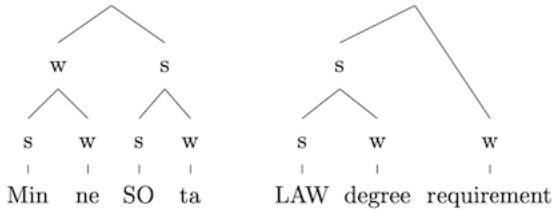
2 Given the below code, answer the following questions:

```
a = [1, 2, [3, 4], 5]
b = a[:]
b[1] = 6
b[2][0] = 7
```

What does b evaluate to?

What does a evaluate to? Are a and b the same? Please explain your reasoning.

3.6 The study of stress is still an open field of inquiry in linguistics—why do we say "alaBAma," but "aLAbama" and "alabaMA" make us cringe? Or how did it come to be that "AMERICAN history professor" and "american HISTORY professor" mean two different things? One model that we use to understand stress actually employs the tree data structure!



In the above diagrams, every node has a "strong" child and a "weak" child, and primary stress is placed on the leaf that has the greatest number of strong parents. In the spirit of computational linguistics, let's write a function that, given one of these tree structures, identifies the stressed part of a word or phrase.[1]

```
def primary_stress(t):
    """
    Returns a STRING corresponding to the stressed part of the word represented by the input tree.
    >>> word = tree("", [
        tree("w", [tree("s", [tree("min")]), tree("w", [tree("ne")])]),
        tree("s", [tree("s", [tree("so")]), tree("w", [tree("ta")])])])
    >>> primary_stress(word)
    'so'
    >>> phrase = tree("", [
        tree("s", [tree("s", [tree("law")]), tree("w", [tree("degree")])]),
        tree("w", [tree("requirement")])])
    >>> primary_stress(phrase)
    'law'
    """
    def helper(t, num_s):
        if is_leaf(t):
            return [label(t), num_s]
        if label(t) == "s":
            num_s = _________________________
        return max([_____________________________________________],
            key = _________________________)
    return _________________________
```

[1] Inspiration for this problem comes from *Liberman, Mark and Alan Prince. 1977. On stress and linguistic rhythm. Linguistic Inquiry. 8:249-336.*, and from the course Linguistics 111 (Phonology).

This page is intentionally left blank.

4 Exam-Level Practice

4.1 **Spring 2018 Midterm 2, Question 5** A *sibling* of a node in a tree is another node that has the same parent.

Implement the function below, which takes a tree t. It returns a list of the labels of all nodes in t that have a sibling. These labels can appear in any order.

```
def siblings(t):
    """Return a list of the labels of all nodes that have siblings in t.
    >>> a = tree(4, [tree(5), tree(6), tree(7, [tree(8)])])
    >>> siblings(tree(1, [tree(3, [a]), tree(9, [tree(10)])]))
    [3, 9, 5, 6, 7]
    """
    result = [_____________________________________________________________________________]

    for b in branches(t):

        _____________________________________________________________________________

    return result
```