Homework 4: Sequences, ADT Trees

hw04.zip (hw04.zip)

Due by 11:59pm on Thursday, October 5

Instructions

Download hw04.zip (hw04.zip). Inside the archive, you will find a file called hw04.py (hw04.py), along with a copy of the `ok` autograder.

Submission: When you are done, submit the assignment by uploading all code files you've edited to Gradescope. You may submit more than once before the deadline; only the final submission will be scored. Check that you have successfully submitted your code on Gradescope. See Lab 0 (/lab/lab00#submitting-the-assignment) for more instructions on submitting assignments.

Using Ok: If you have any questions about using Ok, please refer to this guide. (/articles/using-ok)

Readings: You might find the following references useful:

- Section 2.2 (https://www.composingprograms.com/pages/22-data-abstraction.html)
- Section 2.3 (https://www.composingprograms.com/pages/23-sequences.html#trees)
- Section 2.4 (https://www.composingprograms.com/pages/24-mutable-data.html#sequence-objects)

Grading: Homework is graded based on correctness. Each incorrect problem will decrease the total score by one point. There is a homework recovery policy as stated in the syllabus. **This homework is out of 2 points.**

Required Questions

Getting Started Videos

Sequences

Q1: Filter

Write a function `filter` that takes in a list `lst` and `condition`, a one-argument function that returns either `True` or `False`, and mutates `lst` so that it only contains elements satisfying `condition`.

> *Hint:* Avoid mutating the list as you are iterating through it as this may cause issues with iterating through all the elements.

```
def filter(condition, lst):
    """Filters lst with condition using mutation.
    >>> original_list = [5, -1, 2, 0]
    >>> filter(lambda x: x % 2 == 0, original_list)
    >>> original_list
    [2, 0]
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q filter
```

Q2: Deep Map

Write a function `deep_map_mut` that takes a list `lst` and a one-argument function `func`. `lst` may be a deep list — that is, it may contain other lists. `deep_map_mut` replaces each element of `lst` and its sublists with the result of calling `func` on the element.

`deep_map_mut` does not return the mutated list and does not create any new list objects.

> **Hint:** `type(a) == list` will evaluate to `True` if `a` is a list.

```
def deep_map_mut(func, lst):
    """Deeply maps a function over a list, replacing each item
    in the original list object.
    Does NOT return the mutated list object.

    >>> l = [1, 2, [3, [4], 5], 6]
    >>> deep_map_mut(lambda x: x * x, l)
    >>> l
    [1, 4, [9, [16], 25], 36]
    >>> # Check that you're not making new lists
    >>> s = [3, [1, [4, [1]]]]
    >>> s1 = s[1]
    >>> s2 = s1[1]
    >>> s3 = s2[1]
    >>> deep_map_mut(lambda x: x + 1, s)
    >>> s
    [4, [2, [5, [2]]]]
    >>> s1 is s[1]
    True
    >>> s2 is s1[1]
    True
    >>> s3 is s2[1]
    True
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q deep_map_mut
```

Trees

Q3: Maximum Path Sum

Write a function that takes in a tree and returns the maximum sum of the values along any root-to-leaf path in the tree. A root-to-leaf path is a sequence of nodes starting at the root and proceeding to some leaf of the tree. You can assume the tree will have positive numbers for its labels.

```
def max_path_sum(t):
    """Return the maximum root-to-leaf path sum of a tree.
    >>> t = tree(1, [tree(5, [tree(1), tree(3)]), tree(10)])
    >>> max_path_sum(t) # 1, 10
    11
    >>> t2 = tree(5, [tree(4, [tree(1), tree(3)]), tree(2, [tree(10), tree(3)])])
    >>> max_path_sum(t2) # 5, 2, 10
    17
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q max_path_sum
```

Q4: Has Path

Write a function `has_path` that takes in a tree `t` and a string `word`. Every node in `t` has a label that is one character. `has_path` returns `True` if there is a path in `t` starting from the root where the labels along the path spell out the given `word`. Otherwise, `has_path` returns `False`.

> This data structure is called a trie (https://en.wikipedia.org/wiki/Trie), and it has a lot of cool applications, such as autocomplete!

```
def has_path(t, word):
    """Return whether there is a path in a tree where the entries along the path
    spell out a particular word.

    >>> greetings = tree('h', [tree('i'),
    ...                        tree('e', [tree('l', [tree('l', [tree('o')])]),
    ...                                   tree('y')])])
    >>> print_tree(greetings)
    h
      i
      e
        l
          l
            o
        y
    >>> has_path(greetings, 'h')
    True
    >>> has_path(greetings, 'i')
    False
    >>> has_path(greetings, 'hi')
    True
    >>> has_path(greetings, 'hello')
    True
    >>> has_path(greetings, 'hey')
    True
    >>> has_path(greetings, 'bye')
    False
    >>> has_path(greetings, 'hint')
    False
    """
    assert len(word) > 0, 'no path for empty word.'
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q has_path
```

Check Your Score Locally

You can locally check your score on each question of this assignment by running

```
python3 ok --score
```

This does NOT submit the assignment! When you are satisfied with your score, submit the assignment to Gradescope to receive credit for it.

Submit

Make sure to submit this assignment by uploading any files you've edited **to the appropriate Gradescope assignment.** For a refresher on how to do this, refer to Lab 00 (https://cs61a.org/lab/lab00/#submit-with-gradescope).

Exam Practice

Homework assignments will also contain prior exam-level questions for you to take a look at. These questions have no submission component; feel free to attempt them if you'd like a challenge!

1. Summer 2021 MT Q4: Maximum Exponen-tree-ation (https://cs61a.org/exam/su21/midterm/61a-su21-midterm.pdf#page=10)
2. Summer 2019 MT Q8: Leaf It To Me (https://inst.eecs.berkeley.edu/~cs61a/sp20/exam/su19/mt/61a-su19-mt.pdf#page=9)
3. Summer 2017 MT Q9: Temmie Flakes (https://inst.eecs.berkeley.edu//~cs61a/su17/assets/pdfs/61a-su17-mt.pdf#page=11)