TIPS FOR MIDTERM 2

Computer Science 61A . October 12, 2015 . alvinwan.com/cs61a

This tip sheet does not comprehensively cover *all* types of midterm questions. Instead, it covers the three top types of problems.

01. ERRORS AND MISTAKES

A common midterm theme is to "find the mistake". These can result in errors, doctest failure, or unexpected behavior. Below are a few of the most common errors and mistakes that you will come across both in your code and on the exam.

- Check that the function `return`s the value desired.
- `return` terminates a function.
- `'output'` *with* quotes means the string was *returned* and `output` *without* quotes means the string was *printed*
- Ensure that all mutable objects are modified correctly. This seems more obvious now than it will be, when nested inside of large chunks of code:
    - attributes are modified using `self.`, inside of a method

      ```
      class Obj:
        def method(param):
          x = 5  # x cannot be accessed outside of method
          self.x = 5
      ```

    - lists are modified using `lst[i]` and dictionaries are modified accordingly

      ```
      table = []
      # enumerate(['a', 'b']) = (0, 'a'), (1, 'b')
      for i, table in enumerate(tables):
        table = new_table(x)  # does not modify the room
        tables[i] = new_table(x)
      ```

- Know when mutable objects are shared. (i.e., class attributes are shared across all instances of that class, lists of lists may share component lists)

- Tuples and strings are immutable, whereas lists, dictionaries, and objects are mutable.

02. TREE RECURSION

Just as recursion is divided into three elements, we can divide tree recursion into two. We consider the root - the base case - and an arbitrary node's children - the recursive case. What about the third element of recursion - how the problem is reduced incrementally? In all tree problems, we reduce the problem by traversing to the next level.

Depth-First Traversal Problems

In these types of problems, we rebuild the tree with each element modified. The modification can either be universal (all nodes are changed the same way), or each node is modified based on one of the following variables:

- depth
- node value
- nth-child of its parent
- value of the parent

Note that this list is not comprehensive.

Practice Question 1

```
def make_odd(t):
  """Return a tree where all even-valued nodes are made odd, and
  odd-valued nodes are kept the same

  >>> t = tree(4, [tree(5, [tree(3), tree(2)]), tree(3), tree(8)])
  >>> make_odd(t)
  [5, [[5, [3, 3]], 3, 9]]
  """
  if is_leaf(t):
    return tree(oddify(root(t)))
  return tree(oddify(root(t)), [make_odd(b) for b in branches(t)])

oddify = lambda v: v + 1 if v % 2 == 0 else v
```

Practice Question 2

```
def add_depth(t):
  """Return a tree where the depth of a node is added to the node's
  original value

  >>> t = tree(4, [tree(5, [tree(3), tree(2)]), tree(3), tree(8)])
  >>> add_depth(t)

  """
  def helper(t, depth):
    if is_leaf(t):
      return tree(root(t) + depth)
    return tree(root(t) + depth, [helper(b, depth + 1) for b in
branches(t)])
  return helper(t, 0)
```

Note that some problems *appearing* to be extremely difficult could be reduced to these cases. Also note that on occasion, the base case is not necessary if the only line in your function is a list comprehension.

Breadth-First Traversal Problems

Other Tree Problems

03. BOX-AND-POINTER DIAGRAMS

Cycles

Shared Component Lists