Linked Lists and Mutable Trees

Abhinav Ashar
CS 61A: Structure and Interpretation of Computer Programs

January 18, 2019

1 Linked Lists

a) Similar to regular lists, but each link contains two things: a **value**, and a **pointer** to the next link

b) The last link has a pointer that points to null

c) Linked lists are normally a class in 61A, so you can access the instance variables by doing Link.(instanceVar)

d) You can go through the linked list either recursively or iteratively

e) When solving a function involving a linked list, you might have to change some values. This function can be either destructive (modifies the original list) or non-destructive (does not modify the original list).

Non-destructive: You go through the original linked list A, copy over values that you need to a new linked list B, and edit any values in A according to the problem description by just putting the new value in the same spot in B (the old value still exists in A). Then, return B.

Destructive: You go through the original linked list, and any time you come to a value that needs to be edited in the original linked list, you use dot notation to edit it. No new linked list is created, and you often do not return anything as well.

2 Mutable Trees

Unlike the trees we've previously seen, mutable trees are classes with instance variables rather than with helper functions. This means that the values in each node can be edited using dot notation. branches(t) is equivalent to t.branches and label(t) is equivalent to t.label.