Yet More Trees and Graphs

Christopher R. Genovese

Department of Statistics & Data Science

24 Oct 2024
Session #16

Plan

Climbing Trees

Plan

Climbing Trees

Graphs

Announcements

- `fpc` code, python 3.12
- **Reading**:
  - Review earlier Category Theory readings
  - `DFS.pdf` under `Readings` in the `documents` repo.
- **Homework**: `migit-2` due Tuesday 05 Nov, `kd-tree` Exercises #1 and #2

Plan

Climbing Trees

Graphs

Seeing the Forest for the Trees

There are many, many flavors of tree, but all share a similar structure. Consider:

```
type BinaryTree a = Tip | Branch (BinaryTree a) a (BinaryTree a)

type RoseTree a = Node a (List (RoseTree a))

-- Heterogenous version of Rose Tree, different types on Leaf and Branch
type HTree b l = Leaf l | Branch b (NonEmptyList HTree b l)

type Trie k m a = Trie { data : Maybe a
                       , children : Map k (Trie k m a)
                       , annotation : m      -- m will be a Monoid
                       }

-- One way to think about unrooted trees; there are others.
-- Fin n is the type of integers 1..n. OrderedPairs a is like Pair a a
-- but represents pairs (x, y) where x < y.

type UnrootedTree a = forall n.
         Pair(Injection Fin(n - 1) (OrderedPairs (Fin n)), Fin n -> a)
```

Group work: Building Trees

Using Binary or Rose trees as you prefer, write a quick working implementation of

```
-- Binary case
unfold : (b -> (a, Maybe b, Maybe b)) -> b -> BinaryTree a

-- Rose case
unfold : (b -> (a, List b)) -> b -> RoseTree a
```

The `b` type represents a **seed**. We start with a seed and generate a partial subtree that may include additional seeds from which we generate new subtrees at that position.

Example in the Binary case: `complete_btree(depth)` creates a complete `BinaryTree Int` of the given depth.

Walking and Mapping Trees

A **traversal** of a tree is an organized visit to every node in the tree exactly once.

- Preorder: The root of a subtree is visited before its children.
- Postorder: The children of a subtree are visited before the root.
- Inorder (Binary case): Root-Left-Right order

We often do an ***effectful*** computation at each node.

How much flexibility do we have in the ordering of the nodes? How can you pass information with you as you move along the tree? (Consider how you might number the leaves of a tree by depth.)

Mapping a tree converts a tree to another tree of the same shape. In other words, trees are ***functors***.

Choose one of these and do a quick implementation, using the trees you built. Your function should accept a calculation or action to perform.

Variant: Traverse a nested data structure composed of e.g., vectors, dictionaries, tuples, Iterable objects, et cetera.

Printing Trees

Write a function `to_string` that converts a tree (of whichever form you like) to a printable string that represents the structure of the tree.

Demo

Consider a divide and conquer approach.

- What information do you need to maintain at any point?
- How does recursion figure in to this? How does the recursion reflect the structure of the tree?
- Do you see a Monoid structure here? (You don't have to use it today.)

Class Work: Searching with Tries

Consider a simple form of the "Trie," a special type of tree that can be used for efficiently associating values with a set of strings.

We consider a simplified form here

```
type Trie a = Trie { data : Maybe a
                   , children : Map Char (Trie a)
                   }
```

where `Map k v` is an associative map (e.g., dictionary) from key type `k` to value type `v`.

A string like "foobar" starts at the root node (associated with the empty string) and moves to the child of a node based on successive characters.

We have

```
empty  : Trie a
insert : String -> Trie a -> Trie a
lookup : String -> Trie a -> Maybe a
```

Let's sketch out a trie implementation together.

Discussion challenge: Traversing a tree level by level

Consider a rooted binary or rose tree.

The challenge is:

> *Given a tree with n nodes, create a tree of the same shape but with the leaves numbered* 1..*n level by level from the root and left to right.*

What is the difficulty here? What functions do we need? What are their types? What data structures do we need?

Plan

Climbing Trees

Graphs

Graphs

- Graph representations (adjacency list and matrix)
- Traversals: simple breadth first and depth first
- Getting more out of the traversal.

Let's build it.

The End