Searching Trees & Graphs
Tree-Dichotomy for Recursion

- We often want a different induction/recursion dichotomy for trees than for lists (empty vs. non-empty):
  - The tree is a single node (and is thus a leaf).
  - The tree is a node with offspring (and is thus not a leaf).
Model Independence

- We can abstract away the specific representation being used:
  - \textit{isLeaf(T)} 1 when T is leaf, 0 otherwise.
  - \textit{offspring(T)} the list of offspring of a non-leaf, undefined otherwise

- The implementation depends on which tree model we are using.
Example Implementations

- Labeled-Tree Implementation:
  - `makeTree(Root,ListOfSubTrees) = [Root | ListOfSubTrees];`
  - `isLeaf(T) = rest(T) == [ ];`
  - `getOffspring(T) = rest(T);`
  - `getRoot(T) = first(T);`
Recursion on Trees

- **Basis**: What happens on a single leaf.

- **Induction step**: What happens on a non-leaf.
Example: Height of a Tree

- The height of a tree is the length of the longest path from the root.

- \[ \text{height}(T) = \begin{cases} 0; & \text{isLeaf}(T) \\ 1 + \sum(\text{map}(\text{height}, \text{getOffspring}(T))); \end{cases} \]

- Let recursion do the work for you.
Searching a Labeled Tree

● Suppose we want to find all nodes with labels having a property P.
Searching a Labeled Tree

- findInTree(P, T) =

  Root =getRoot(T),

  foundInRest =
      mappend((S)=>findInTree(P, S), getOffspring(T)),

  P(Root) ? [Root | foundInRest] : foundInRest;
Depth-First Search

- The preceding expresses only one form of search:

  depth-first search

  The pattern is to search “deeper” before “broader”.
Depth- vs. Breadth- First

Example: Find evens
Breadth-First: Wavefront Analogy

1

2

3 6 5

4 9

8

7

1.
2.
3.
4.
5.
6.
7.
8.
9.
Advantage of Breadth-First
Breadth-First Searching a Labeled Tree

- Easier if we generalize to search a forest:

- `findInTreeBF(P, T) = findInForest(P, [T]);`

- `findInForest(P, [ ]) => [ ];`  
  
  forest of one tree

- `findInForest(P, [Tree | Trees]) =>`  
  
  `Root = root(Tree),`  
  
  `foundInRest = findInForest(P, append(Trees, getOffspring(Tree))),`  
  
  `P(Root) ? [Root | foundInRest] : foundInRest;`
Searching Graphs vs. Trees

- Basic ideas still apply, but
  - In graph, avoid re-searching same nodes due to fan-in
  - In graph, avoid infinite loops.
  - How?
Depth- vs. Breadth-First in Graph

Example: Find evens
Searching Without Recursion

- Depth-First: Use Stack
- Breadth-First: Use Queue
- Avoid Fan-in and Cycles:
  - “Mark” nodes as encountered
    - Refuse to re-search from a marked node
  - Marking can be metaphoric, e.g. by membership on a list, or
    The node itself can be marked (non-functional programming).
A maze is an implicit graph

Nodes are identifiable by position

The arcs are implicit, determined by adjacent spatial positions.

Marking can be done in a “parallel array”
Recovering Path

- Searching for the first element satisfying some property.
- Want not just the element, but the path from the starting point to the element.
- How to accomplish?
Lineage and Back Pointers

- Non-root nodes are encountered from a parent node.
- When a node is encountered, could keep a record of the parent.
- Keep a list of nodes during descent is one means.
- Setting a “back pointer” to the parent is another; could use an association list of [child, parent] for example.
Two Birds with ...

- If we use back-pointers to remember parents, we don’t also need marking:
  - A node with a known parent is implicitly marked.
  - A node without a known parent is not marked.
Non-functional Back-Pointers

- When not restricted to functional programming techniques only, can use destructive setting of back-pointers for greater efficiency.