Massachusetts Institute of Technology
6.042J/18.062J, Fall '02: Mathematics for Computer Science
Professor Albert Meyer and Dr. Radhika Nagpal

In-Class Problems — Week 6, Fri

1 Problems

Problem 1. [carried over from Monday, Oct. 7]

The definition of Recursive Ordered Binary Trees, RecBinT, from Week 6 Notes is repeated in an Appendix.

(a) Give a recursive definition of the set of leaves of a RecBinT.

Full Binary Trees, FullBinT, are the special case of RecBinT's in which only the makeboth constructor is used. For example, Figure 1 shows an FBT with 13 nodes of which 7 are leaves:

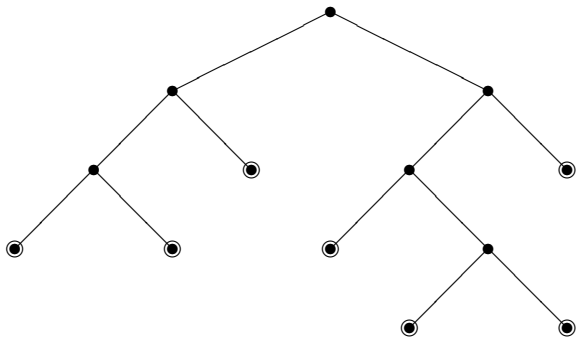

Figure 1: A full binary tree.

(b) Prove by structural induction on the definition of FullBinT that for all FullBinT's, t,

$$2\,|\text{leaves}(t)| = |\text{nodes}(t)| + 1.$$

Problem 2. [carried over from Monday, Oct. 7]

The Elementary 18.01 Functions (F18's) are the set of functions of one real variable defined recursively as follows:

1. The identity function, $\mathrm{id}(x) ::= x$ is an F18,
2. any the constant function is an F18,
3. $\sin, \cos, e^x$ are F18's,
   and if f, g are F18's, then so are
4. $f + g$, $f - g$, fg, f/g
5. the inverse function f^{-1},
6. the composition $f \circ g$.

Show that the Elementary 18.01 Functions are *closed under taking derivatives*. That is, show that if f is an F18, then so is df/dx.

Problem 3. We didn't get to this problem on Friday, so we'll do it on the pset.

We can generalize win-lose 2-player terminating games of perfect information to games with "payoff" amounts. In these games, two players called the *max-player* and the *min-player* alternate moves until the game ends with the min-player paying some payoff amount to the max-player. How much the min-player pays depends on how the game ends. Negative payoffs mean the max-player pays the min-player. The max-player moves first.

Such games are defined by finite-path trees with leaves labelled with real numbers. These are the payoff amounts. The max-player tries to arrive at a leaf with as large a payoff as possible, and the min-player tries to minimize the payoff to the max-player.

Definition. The set of payoff-game trees, PayT, can be defined recursively as follows:

1. If T is a graph with one vertex, v, and no edges, then T is a PayT and $\mathrm{root}(T) ::= v$.
2. if $\mathcal{S}$ is a set of PayT's such that no vertex occurs in more than one tree in $\mathcal{S}$, and v is a "new" element that is not a vertex of any tree in $\mathcal{S}$, then T is in PayT where $\mathrm{root}(T) = v$ and the edges of T are the edges of all the trees in $\mathcal{S}$ along with edges connecting $\mathrm{root}(T)$ to the roots of each of the trees in $\mathcal{S}$. The trees in $\mathcal{S}$ are called the *children* of T.

We define functions $\text{max-value}(T)$ and $\text{min-value}(T)$ on payoff-game trees, $T \in \text{PayT}$, recursively on the definition of PayT:

1. If T is a single node labelled r, then
$$\text{max-value}(T) = \text{min-value}(T) ::= r.$$
2. If the nonempty set $\mathcal{S}$ is the set of children of T, then
$$\begin{aligned} \text{max-value}(T) &::= \text{lub}\,\{\text{min-value}(S) \mid S \in \mathcal{S}\} \\ \text{min-value}(T) &::= \text{glb}\,\{\text{max-value}(S) \mid S \in \mathcal{S}\}\,. \end{aligned}$$

(a) Suppose a payoff-game tree, T, is *finite*. Prove that

1. If the max-player is the first player to move in T, then he has a strategy that guarantees his payoff will be at least max-value(T), no matter how the min-player behaves.
2. If the max-player is the second player to move in T, then he has a strategy that guarantees his payoff will be at least min-value(T).
3. Likewise, if the min-player is the first player to move in T, then she has a strategy that guarantees the payoff to max-player will be at most min-value(T).
4. If the min-player is the second player to move in T, then she has a strategy that guarantees the payoff to max-player will be at most max-value(T).

(So the players may as well skip playing and just have the min-player pay max-value(T) to the max-player.)

(b) Now generalize the previous part to arbitrary PayT's. *Hint:* It might be helpful to assume the payoff amounts at the leaves are bounded above and below by particular numbers. After settling this case, try it without assuming bounds. Note that in the unbounded case, max-value T may be $+\infty$ and min-value T may be $-\infty$.

2 Appendix: From Week 6 Notes

Several basic examples of recursively defined data types are based on *rooted trees*. These are possibly infinite directed trees, $T = (V_T, E_T)$, with a necessarily unique "root" vertex root(T), such that every vertex is reachable by a directed path from the root. Finite-Path Trees from Week 5 Notes, for example, are the class of rooted trees which do not have any infinite directed path from the root.

Another important class of trees are the *ordered binary trees*. These are possibly infinite rooted trees with labelled edges, such that at most two edges leave each vertex. If two edges leave a vertex, one is labelled `left` and the other is labelled `right`. If one edge leaves a vertex, it is labelled either `left` or `right`.

Definition 3.1. We will define a special class of ordered binary trees called the *recursive ordered binary trees*, RecBinT:

1. If G is a graph with one vertex and no edges, then G is a RecBinT. That is, $(\{v\}, \emptyset) \in$ RecBinT and root$((\{v\}, \emptyset)) = v$.
2. If $T = (V, E)$ is in RecBinT, and $\mathbf{n}$ is a "new" node not in V, then the graph, makeleft(T), made by adding an edge labelled `left` from $\mathbf{n}$ to root(T) is a RecBinT. That is,

$$\text{makeleft}(T) ::= (V \cup \{\mathbf{n}\}, E \cup \{(\mathbf{n}, \text{root}(T), \texttt{left})\}) \in \text{RecBinT},$$

   where (v, w, ℓ) is the directed edge from vertex v to vertex w with label ℓ.
3. Same as above, with "right" in place of "left."
4. If $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ are in RecBinT, V_1 and V_2 are disjoint, and $\mathbf{n}$ is a "new" node not in $V_1 \cup V_2$, then the graph, makeboth(T_1, T_2), made by adding an edge labelled `left` from $\mathbf{n}$ to root(T_1) and an edge labelled `right` from $\mathbf{n}$ to root(T_2) is a RecBinT. That is,

$$\text{makeboth}(T_1, T_2) ::= (V_1 \cup V_2 \cup \{\mathbf{n}\}, E_1 \cup E_2 \cup \{(\mathbf{n}, \text{root}(T_1), \texttt{left}), (\mathbf{n}, \text{root}(T_2), \texttt{right})\}) \in \text{RecBinT}.$$

These cases are illustrated in Figure 2, with edges labelled "left" shown going down to the left, and edges labelled "right" shown going down to the right.

Note that RecBinT is precisely the set of *finite* ordered binary trees.

A special case of RecBinT's are the *full* ordered binary trees, FullBinT. These are RecBinT's in which every node is either a *leaf* (*i.e.*, has out-degree zero) or has both a left and right subtree (see Figure 1). In other words rules 2. and 3. in the definition of RecBinT are not used in defining the subset FullBinT $\subset$ RecBinT.

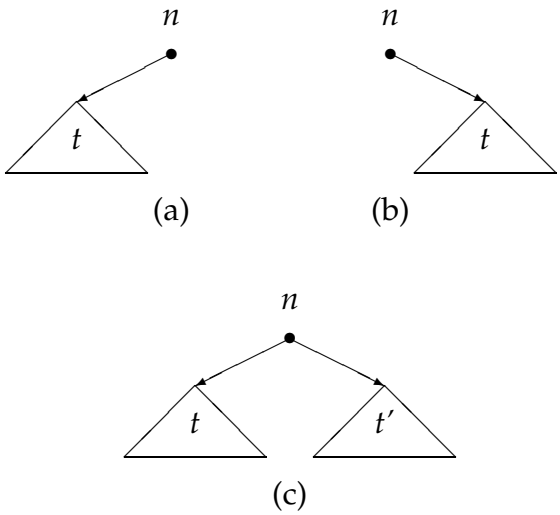

Figure 2: Building a binary tree: (a) makeleft(T), (b) makeright(T), and (c) makeboth(T_1, T_2).